

AD-A193 429

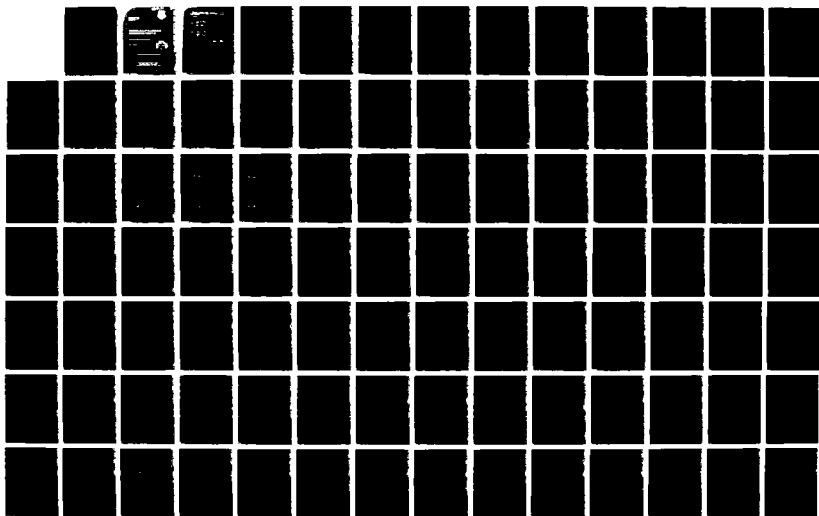
SOFTWARE QUALITY MEASUREMENT DEMONSTRATION PROJECT (I)
(U) COMPUTER SCIENCES INNOVATIONS PALM BAY FL
J L WARTHMAN DEC 87 RADC-TR-87-247 F30602-85-C-0180

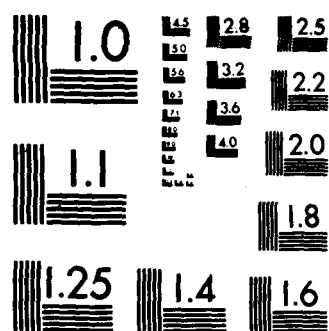
1/2

UNCLASSIFIED

F/G 12/5

NL





G MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A193 429

AD-A193 429
Final Report
1987-1987

SOFTWARE QUALITY MEASUREMENT DEMONSTRATION PROJECT (I)

Computer Science Innovations, Inc.

James L. Warthman

DTIC
ELECTE
APR 06 1988
S D
CH

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

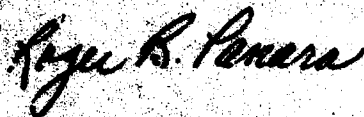
ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

88 4 5 033

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

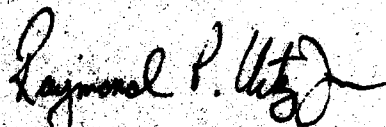
RADC-TR-87-247 has been reviewed and is approved for publication.

APPROVED:



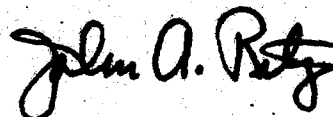
ROGER B. PANARA
Project Engineer

APPROVED:



RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER:



JOHN A. RITZ
Directorate of Plans & Programs

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 (COEE) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188		
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS N/A			
2a. SECURITY CLASSIFICATION AUTHORITY N/A			3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.			
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A						
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A			5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-87-247			
6a. NAME OF PERFORMING ORGANIZATION Computer Science Innovations, Inc.		6b. OFFICE SYMBOL (If applicable)		7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (COEE)		
6c. ADDRESS (City, State, and ZIP Code) 1280 Clearmont Street N.E. Palm Bay FL 32905			7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700			
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Rome Air Development Center		8b. OFFICE SYMBOL (If applicable) COEE		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F30602-85-C-0180		
8c. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700			10. SOURCE OF FUNDING NUMBERS			
			PROGRAM ELEMENT NO 63728F	PROJECT NO 2527	TASK NO 03	WORK UNIT ACCESSION NO. 07
11. TITLE (Include Security Classification) SOFTWARE QUALITY MEASUREMENT DEMONSTRATION PROJECT (I)						
12. PERSONAL AUTHOR(S) James L. Warthman						
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM Aug 85 TO Apr 87		14. DATE OF REPORT (Year, Month, Day) December 1987		
15. PAGE COUNT 116						
16. SUPPLEMENTARY NOTATION N/A						
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Software Quality Software Quality Metrics			
FIELD	GROUP	SUB-GROUP				
12	05					
19. ABSTRACT (Continue on reverse if necessary and identify by block number) The purpose of this project was to assess the feasibility and utility of transitioning the software quality specification and evaluation methodology found in the Specification of Software Quality Attributes Guidebooks (3 vols., RADC-TR-85-37, Feb 85) to the software acquisition environment. Two decision aid developments were used as the test programs. Specific recommendations have been made to improve on the methodology, thereby enhancing its acceptance by and utility to software acquisition managers.						
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS				21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL ROGER B. PANARA				22b. TELEPHONE (Include Area Code) (315) 330-3655		
				22c. OFFICE SYMBOL RADC (COEE)		

DD Form 1473, JUN 86

Previous editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

PREFACE

This document is the final technical report (CDRL A004) for the Software Quality Measurement Demonstration contract, number F30602-85-C-0180. The contract was performed for Rome Air Development Center (RADC) to evaluate the RADC software quality framework.

This report is organized into four major sections as follows:

- I. EXECUTIVE SUMMARY
- II. APPROACH
- III. FINDINGS & RECOMMENDATIONS
- IV. CONCLUSIONS

Section I provides background on RADC's software quality framework and defines the objectives of this study. Section II details our approach to the evaluation of the framework. Section III examines our findings, and suggests recommendations for enhancements to the methodology and the framework. Section IV summarizes our conclusions.

This study is based on earlier efforts to define the software quality framework. In particular, the Boeing Aerospace Company produced a set of documents (guidebooks) for use in implementing the methodology. The guidebooks were published for RADC under contract number F30602-82-C-0137.

I would like to acknowledge the efforts of two colleagues, Tod R. Loebel and Carolyn D. Midwood for their significant contributions to this study. Many of the specific findings and recommendations are a direct result of their painstaking study of the framework and its application.



Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
R-1	

TABLE OF CONTENTS

1.0	EXECUTIVE SUMMARY	1
1.1	Objectives	1
1.2	Overview of Contract Results	1
1.3	Background	4
1.4	Senior Battle Staff Decision Aids	6
2.0	APPROACH	7
2.1	Software Quality Specification	7
2.1.1	Identify Functions	8
2.1.2	Assign Initial Quality Goals	8
2.1.3	Consider Interrelationships	12
2.1.4	Consider Costs	14
2.1.5	Specify Criteria	14
2.1.6	Specify Metrics	15
2.1.7	Assess Compliance With Requirements	15
2.1.8	Publish Software Quality Requirements Report	15
2.2	Software Quality Evaluation	15
2.2.1	Identify Allocation Relationships	16
2.2.2	Apply Worksheets	16
2.2.2.1	Gather Source Material	16
2.2.2.2	Metric Element Evaluation Packages	16
2.2.2.3	Answer Worksheet Questions	17
2.2.3	Score Factors	18
2.2.4	Analyze Scoring	21
2.2.5	Recommend Corrective Action	21
3.0	FINDINGS AND RECOMMENDATIONS	22
3.1	Software Quality Framework	22
3.1.1	Software Quality Factors	22
3.1.1.1	Efficiency	22
3.1.1.2	Integrity	24
3.1.1.3	Reliability	25
3.1.1.4	Survivability	25

TABLE OF CONTENTS

3.1.1.5	Usability	26
3.1.1.6	Correctness	26
3.1.1.7	Maintainability	27
3.1.1.8	Verifiability	27
3.1.1.9	Expandability	28
3.1.1.10	Flexibility	28
3.1.1.11	Interoperability	29
3.1.1.12	Portability	30
3.1.1.13	Reusability	30
3.1.1.14	Other Candidate Factors	30
3.1.1.14.1	Trustworthiness	31
3.1.2	Criteria & Metrics	31
3.1.2.1	Accuracy	33
3.1.2.2	Anomaly Management	33
3.1.2.3	Application Independence	34
3.1.2.4	Augmentability	35
3.1.2.5	Autonomy	35
3.1.2.6	Commonality	36
3.1.2.7	Completeness	36
3.1.2.8	Consistency	36
3.1.2.9	Distributedness	37
3.1.2.10	Document Accessibility	37
3.1.2.11	Effectiveness-Communication	37
3.1.2.12	Effectiveness-Processing	37
3.1.2.13	Effectiveness-Storage	38
3.1.2.14	Functional Overlap	38
3.1.2.15	Functional Scope	38
3.1.2.16	Generality	39
3.1.2.17	Independence	40
3.1.2.18	Modularity	40
3.1.2.19	Operability	40
3.1.2.20	Reconfigurability	41
3.1.2.21	Self-Descriptiveness	41

TABLE OF CONTENTS

3.1.2.22	Simplicity	42
3.1.2.23	System Accessibility	42
3.1.2.24	System Clarity	42
3.1.2.25	System Compatibility	43
3.1.2.26	Traceability	43
3.1.2.27	Training	44
3.1.2.28	Virtuality	44
3.1.2.29	Visibility	45
3.2	Software Quality Specification	45
3.2.0	Software Quality Specification Methodology	45
3.2.1	Select and Specify Quality Factors	46
3.2.1.1	Identify Functions (Step 1)	46
3.2.1.2	Assign Quality Factors and Goals (Step 2)	46
3.2.1.2.1	Command and Control Quality Concerns	50
3.2.1.2.2	System Quality Factors	50
3.2.1.2.3	Quality Requirements Survey	50
3.2.1.2.4	Complimentary Quality Factors	51
3.2.1.2.5	Quality Goals Assignment	51
3.2.1.3	Consider Interrelationships (Step 3)	52
3.2.1.3.1	Shared Criteria	52
3.2.1.3.2	Beneficial and Adverse Relationships	53
3.2.1.3.3	Quantification of Relationships	53
3.2.1.3.4	Review of Quality Goals	54
3.2.1.4	Consider Costs (Step 4)	55
3.2.1.4.1	Life-Cycle Quality Costs and Benefits	55
3.2.1.4.2	Cost Variation Estimates	56
3.2.1.4.3	Cost Effects of Factor Interrelationships	57
3.2.1.4.4	Review of Quality Goals	57
3.2.2	Select and Specify Quality Criteria	58
3.2.2.1	Select Criteria (Step 1)	58
3.2.2.2	Assign Weighting Formulas (Step 2)	58
3.2.2.3	Consider Interrelationships (Step 3)	59

TABLE OF CONTENTS

3.2.3	Select and Qualify Metrics	60
3.2.3.1	Identify Metrics (Step 1)	60
3.2.3.2	Select and Qualify Metric Elements (Step 2)	61
3.2.4	Assess Compliance with Requirements	62
3.2.4.1	Review Requirements Allocations and Evaluation Formulas	62
3.2.4.2	Review Factor Scores	62
3.2.4.3	Review Criteria Scores	63
3.2.4.4	Review Metric Scores	63
3.2.A	Appendix A - Metric Worksheets	63
3.2.B	Appendix B - Factor Scoresheets	63
3.2.C	Appendix C - Software Quality Evaluation Report	64
3.3	Software Quality Evaluation	64
3.3.0	Software Quality Evaluation Methodology	64
3.3.1	Identify Allocation Relationships	64
3.3.2	Apply Worksheets	65
3.3.2.1	Prepare Worksheets (Step 1)	65
3.3.2.2	Gather Source Material (Step 2)	66
3.3.2.3	Answer Worksheet Questions (Step 3)	66
3.3.3	Score Factors	68
3.3.3.1	Prepare Scoresheets (Step 1)	68
3.3.3.2	Calculate Factor Scores (Step 2)	75
3.3.4	Analyze Scoring	75
3.3.4.1	Calculate Functional Scores (Step 1)	75
3.3.4.2	Compute Scoring Trends (Step 2)	75
3.3.4.3	Compare Scores with Requirements (Step 3)	76
3.3.4.4	Analyze Variations (Step 4)	76
3.3.5	Recommend Corrective Actions	76
3.3.5.1	Summarize Problems (Step 1)	76
3.3.5.2	Provide Recommendations (Step 2)	77

TABLE OF CONTENTS

3.3.6	Automation	77
3.3.A	Appendix A - Metric Worksheets	77
3.3.B	Appendix B - Factor Scoresheets	77
3.3.C	Appendix C - Software Quality Evaluation Report	77
4.0	CONCLUSIONS	79
APPENDIX A: BIBLIOGRAPHY		A-1
APPENDIX B: GLOSSARY		B-1
APPENDIX C: QUALITY REQUIREMENTS SURVEY RESPONSES		C-1
APPENDIX D: SAMPLE QUALITY EVALUATION FORMS		D-1
APPENDIX E: SOFTWARE QUALITY REQUIREMENTS REPORT		E-1

LIST OF FIGURES

FIGURE	TITLE	PAGE
2.1.2	Sample Quality Requirements Survey	10
2.2.3-1	RAA"Develop Apportionment Recommendations"Factor Goals & Scores	18
2.2.3-2	RAA"Provide Rationale for Recommendations"Factor Goals and Scores	19
2.2.3-3	EPAA"Develop Knowledge"Factor Goals and Scores	19
2.2.3-4	EPAA "Monitor Enemy Performanc" Factor Goals and Scores	20
2.2.3-5	EPAA "Analyze Enemy Performance" Factor Goals and Scores	20
3.1.2	Revised Framework	32
3.2.1.2-1	Acquisition Hierarchy	48
3.2.1.2-2	Sample Acquisition Attributes	49
3.3.3.1-1	Criteria Scoresheet	71
3.3.3.1-2	Function Factor Scoresheet	72
3.3.3.1-3	Function Criteria Scoresheet	73
3.3.3.1-4	Scoring Process Overview	74

LIST OF TABLES

TABLE	TITLE	PAGE
2.1.2	Resulting Initial Quality Factor Goals	11
2.1.3	Final Quality Factor Goals	13
2.1.5	Criteria Weighting Formulas for EPAA Develop Knowledge Function	14
2.2.2	Time to Apply Worksheets	18

SECTION 1 EXECUTIVE SUMMARY

This technical report presents results of the Software Quality Measurement Demonstration (SQMD) study. This work was performed for Rome Air Development Center (RADC) by Computer Science Innovations, Inc. (CSI) under Contract No. F30602-85-C-0180.

1.1 OBJECTIVES

The goals of this 18 month effort were to evaluate RADC's software quality measurement framework and to validate metrics for adequacy and correctness. Conclusions were drawn regarding the utility of the methodology as a quantitative input to the overall quality assurance process, and specific recommendations were made to improve the methodology and the metrics [1].

1.2 OVERVIEW OF CONTRACT RESULTS

CSI applied the RADC software quality measurement framework to two Senior Battle Staff Decision Aids (SBSDA) which were developed by PAR Technology Corporation under contract to RADC. We followed the Software Quality Guidebooks, Volumes I and II, to perform both the quality specification and quality evaluation tasks. In this respect, we took on the roles of a software acquisition manager (SAM) and of an independent quality assurance team. The experience gained was used to evaluate the methodology and elements of the framework. During this process we conducted several Technical Interchange Meetings (TIMs) with various RADC personnel and other contractors working with the software quality program.

We found the software quality framework to be useful in the context of the software acquisition process. By focusing on software quality from the beginning, the SAM and the developer put much needed emphasis on quality concerns such as MAINTAINABILITY, RELIABILITY, PORTABILITY, REUSABILITY, and USABILITY.

The guidebooks and framework are related to DoD-STD-2167 (also referred to herein as 2167). This software development standard requires that software quality be specified, but provides no guidance for doing so. The RADC software quality program provides the means to both specify software quality goals and measure, periodically, the progress toward those goals.

Application of software quality measurement during development is particularly beneficial because quality deficiencies can be detected and corrected early. Savings which result can be expected to more

than compensate for the additional cost of specifying and evaluating software quality.

The software quality framework consists of a multi-tiered structure of quality attributes. This organization is very effective, since it lends itself to evolution and tailoring. In fact, the framework *must* evolve in order to adapt to advances in software engineering. Our study revealed several areas in which key technologies are not yet accounted for by the metrics. For example, the framework should account for the use of robust transaction processing techniques, fault-tolerant architectures, knowledge based concepts, and fourth generation languages. In addition to advances in the framework, the methodology as documented in the guidebooks must evolve as experience is gained in its use. We have noted several areas which should be modified to improve the methodology and, thus, increase its acceptance by SAMs. For example, the procedures involved with computing and analyzing factor scores should be clarified.

As a result of this study we have proposed a number of significant changes to the framework. Four factors, because they are more appropriately specified with *functional and performance requirements*, should be deleted. These factors are EFFICIENCY, INTEGRITY, EXPANDABILITY, and INTEROPERABILITY. We felt strongly that these factors, while clearly important to the software acquisition process, are not appropriately specified and evaluated as quality factors. If these attributes are needed then they should be specified concretely as functional, performance, and interface requirements. An additional factor, TRUSTWORTHINESS, should be incorporated into the framework. This factor is related to RELIABILITY, but differs in the way it accounts for the severity of possible failures.

Several changes to the criteria are in order. In particular, the concept of complimentary factors should be eliminated. Complimentarity, as discussed in Volume II of the Boeing Guidebooks, refers to the fact that certain factors should not be specified or evaluated by themselves. For example, if RELIABILITY is important, then CORRECTNESS and VERIFIABILITY also are important. The Guidebook states that "even if the metric scores were high for RELIABILITY, but the software was incorrect or difficult to verify, the *actual* reliability could be low..." (emphasis added). The implication is that the factor RELIABILITY does not measure some software characteristics which are, in fact, important to the factor. We believe that each factor definition must incorporate all the attributes, in the form of criteria, which are needed for a *complete* definition of the factor. In that way we can be sure that the specification and evaluation of any *specific* factor can stand alone. We should not need to consider the values of other (complimentary) factors to determine if, indeed, the quality of interest has been successfully achieved. Only by eliminating the concept of complementary factors can these goals be realized.

We have also identified a number of changes to metrics and metric elements. These recommendations can be classified generally as clarification of questions, modifications to metric computations, and use of metrics during different life cycle phases.

Aside from the framework modifications mentioned above, we have made a number of general recommendations. First, the collection of metric data is time consuming, and somewhat subjective. Automation will help greatly in improving the efficiency and objectivity of metric data collection. RADC's Quality and Productivity Tool (QPT) will be instrumental in expanding the automation of the software quality process.

We have raised a fundamental concern about the meaning of numeric scores - factor, criteria, and metric. At present, no statistically valid basis has been developed which would allow factor goals to be compared with factor scores. Neither can scores from one worksheet be compared with scores from other worksheets. Finally, scores for one factor cannot be meaningfully compared with scores from other factors. The guidebooks should explain this limitation, at least until such a statistical basis has been developed.

We found that the guidebooks are useful to those who are experienced with quality measurement technology. However, the primary audience for the guidebooks - SAMs and software developers - do not generally have the required background to properly implement the methodology. This problem could be minimized by some restructuring and clarification of the guidebooks. We recommend that Volumes II and III be rewritten to focus on the detailed steps required to implement the methodology. These instructions should not be confused by interspersing examples and theory. These volumes should contain more detailed, "cookbook like" instructions as well as a complete set of blank forms which could be copied and used at each stage in the process. A third, tutorial volume should also be prepared. This book would contain the history, theory and examples which are scattered through the current Volumes II and III. It also would contain expanded, more detailed examples, both simple and complex. By such a restructuring, the methodology would become much more accessible to the SAMs and software developers who must use it.

Finally, we believe that users must understand both the strengths and weaknesses of the quality measurement technology. Specifically, since it hasn't yet been validated, scoring analysis should not be emphasized yet. Instead, users should be encouraged to use scores to identify weak areas during software development. This significant benefit of the methodology can be exploited by looking for anomalies in unit, CSC, and CSCI scores. A criterion score that is much lower than average for a given system deserves detailed study.

The remainder of this report provides the details of our approach, our findings, and recommendations. We hope that our contribution to quality measurement technology is a positive one, and will be used to benefit this important tool of software acquisition managers and software developers.

1.3 BACKGROUND

Software Quality Assurance was, at one time, primarily an exercise of testing to insure that the product satisfied the requirements. It is now generally acknowledged that this type of "QA" is inadequate because quality must be built in, not tested in. The traditional approach typically resulted in systems that were expensive to modify and maintain, and often did not meet the needs of operational personnel.

The need for a quantitative way to specify and measure software quality was recognized, and in 1976 RADC began exploring software quality measurement. At that time, an effort was undertaken to explore methods of specifying and measuring software quality in a quantifiable manner (contract no. F30602-76-C-0417, Factors in Software Quality). An important goal was to make this technology available to Air Force acquisition managers. This study defined software quality in a hierarchical fashion. At the top of this framework were eleven quality factors which represented user-oriented views of quality. These factors were CORRECTNESS, EFFICIENCY, INTEGRITY, USABILITY, TESTABILITY, FLEXIBILITY, REUSABILITY, MAINTAINABILITY, RELIABILITY, PORTABILITY, and INTEROPERABILITY. Each factor was described in terms of software oriented attributes called criteria. Examples of criteria are TRACEABILITY, CONSISTENCY, COMPLETENESS, MODULARITY, and MACHINE INDEPENDENCE. Each criterion was further defined in terms of metrics, which were quantitative measures of an attribute. Metrics were items such as quantity of comments, completeness checklist, and complexity measure. This entire framework would then be used to specify, measure, and predict the quality of a software system at various points in its life cycle. This initial effort resulted in publication of the Preliminary Handbook on Software Quality for an Acquisition Manager (RADC-TR-77-369) [2].

In 1978 RADC sponsored a study (F30602-78-C-0216, Software Quality Metrics Enhancement) which involved studying the applicability of the framework to software developments other than Command and Control, and validating and refining several specific metrics. It was found that quality measurement could be used effectively within a Management Information System (MIS) environment. Several metrics were deleted for various reasons, and several were added. A sensitivity analysis was used and demonstrated to be an effective tool, based on quantitative quality measures. Finally, a new Software Quality Measurement Manual (RADC-TR-80-109) was produced for acquisition managers [3].

In 1979, RADC awarded a contract (F30602-79-C-0267) to automate the collection of metric data. The Automated Measurement Tool (AMT) was developed and delivered to the Air Force in 1981.

RADC sponsored an effort starting in 1980 (F30602-80-C-0265 - Software Interoperability and Reusability) to further enhance the software quality framework. Two quality factors, INTEROPERABILITY and REUSABILITY, were refined by the addition of new criteria and metrics, and their applicability was validated. A new Guidebook for Software Quality Measurement (RADC-TR-83-174) was produced [4].

Also in 1980, RADC sponsored research (F30602-80-C-0330 - Software Quality Measurements for Distributed Systems) to extend the software quality framework into systems with distributed software. As a result, modifications were made to the quality framework. Two new quality factors, SURVIVABILITY and EXPANDABILITY, were introduced. This was documented in Software Quality Measurement for Distributed Systems (RADC-TR-83-175) [5].

In 1982, RADC sponsored research (F30602-82-C-017, Specification of Software Quality Attributes) which was intended to consolidate previous research and develop the methodology for use by an Air Force software acquisition manager. This involved, in part, adopting the terminology of DoD-STD-2167 (DoD-STD-SDS). A three volume report was produced (RADC-TR-85-37), two volumes of which provide the software acquisition manager with guidelines for the specification and evaluation of software quality [6]. Also, a plan was prepared to validate the software quality framework.

At present, several RADC sponsored software quality activities are progressing in parallel. The Automated Measurement System (AMS) contract is being conducted to refine and expand the AMT. Another tool, the Environment Measurement Instrumentation (EMI), is being developed to support data collection within a life cycle software engineering environment. The AMS is about to evolve into the Quality and Productivity Tool (QPT) which will integrate metric tools into a more comprehensive software engineering environment.

This contract (F30602-85-C-0180, Software Quality Measurement Demonstration I) was conducted to evaluate the methodology and validate the metrics which have been developed and refined under the contracts previously referenced. The target projects for CSI's study were two decision aid programs developed under the Senior Battle Staff Decision Aids (SBSDA) project. A parallel study (Software Quality Measurement Demonstration II) was conducted independently against two other decision aid software systems which were also developed under the SBSDA project.

1.4 SENIOR BATTLE STAFF DECISION AIDS

This subsection contains a brief overview of the SBSDA system. RADC sponsored this development effort with PAR Technology Corporation as prime contractor. PAR's subcontractors were Betac Corporation, Perceptronics Inc., and Knowledge Systems Concepts, Inc.

The SBSDA system consists of four prototype decision aids [7,8]. The aids were designed to demonstrate that Artificial Intelligence, Decision Analysis, and Operations Research can assist Tactical Air Force Senior Battle Staff decision makers. The aids developed were:

1. Resource Apportionment Aid (RAA),
2. Enemy Performance Assessment Aid (EPAA),
3. Enemy Sortie Capability Measurement Aid (ESCMA), and,
4. Enemy Course of Action Evaluation Air (ECOAEA)

CSI is studying the RAA and EPAA aids under this study.

The primary use of the Resource Apportionment Aid (RAA) is to support the Tactical Air Forces (TAF) Commander's Air Apportionment Conference. This includes recommendations for the most efficient use of air power, rationale for this apportionment, and how the apportionment effort directly supports the Joint Task Force (JTF) commander's guidance.

The Enemy Performance Assessment Aid (EPAA) is intended to be used by an air component command headquarters staff or the intelligence staff of a tactical air force. Its function is to assess the current state of an enemy's combat capability.

SECTION 2 APPROACH

This section describes how RADC's methodology was applied to the SBSDA. Also, it identifies the procedures that were used to collect information for evaluating the guidebooks and the framework.

This section documents the procedures we followed. Highlights are presented, along with considerations which were unique to the SBSDA. The RAA and EPAA Software Quality Requirements Reports [9,10], and the Implementation and Validation Plan (CDRL A002) [11] further define the procedures which were followed.

This section is organized into two main topics. First, the process used to specify software quality for the EPAA and RAA is described. Second, the quality evaluation process is described. Unless otherwise specified, these descriptions apply to both EPAA and RAA.

2.1 SOFTWARE QUALITY SPECIFICATION

This subsection describes, at a high level, the process of specifying software quality for the decision aids. In general, quality specification was performed in accordance with the instructions in the Software Quality Specification Guidebook, Volume II. Exceptions to this are noted below.

First, functions were identified and quality factor goals were defined. Next, criteria goals and weighting formulas were determined for these functions. Finally, the metrics and metric elements were examined for applicability to these functions.

Since the SBSDA program was completed in September 1985, the methodology was applied "outside" the development life cycle context. The guidebooks briefly discuss this case and how it differs from the more typical case. Ideally, however, the study of the methodology and framework should be performed against a project during its development. The quality requirements could then be incorporated into the System / Segment Specification and allocated down into design documents. Quality could be evaluated at each major review, and necessary adjustments made. Such a scenario would be a more representative application of the methodology, and would allow more conclusions to be drawn regarding the validity of the framework. In particular, this approach would eliminate the potential for bias which existed in our study, due to the unavoidable hindsight of specifying and evaluating quality for an already completed system.

The software quality framework has been designed to work with products of the software development process, such as documentation and code. This framework assumes that the development is done according to DoD-STD-2167 [12], and the guidebooks describe procedures for collecting quality metric data from the various DID formats which this standard specifies. The SBSDA contract was awarded prior to implementation of the standard; therefore, it was necessary to develop a mapping between the SBSDA documents and 2167. It is felt that, because the SBSDA effort was conducted as an exploratory development and emphasis was placed on getting the decision aids to perform operations which were previously labor intensive, the software documentation was incomplete. Despite this, the findings and recommendations regarding the software quality framework and methodology are valid. The quality of SBSDA documentation did not affect our conclusions.

2.1.1 Identify Functions

The first step in performing quality specifications was to identify the major software functions for each decision aid. This task was particularly difficult, due to the structure of the SBSDA documentation. RAA, in particular, was challenging because several different functional decompositions were documented in various parts of the documentation.

PAR assisted with this task, and we arrived at a reasonable set of functions for the two systems:

- EPAA Develop Knowledge
 Monitor Enemy Performance
 Analyze Enemy Performance

- RAA Develop & Analyze Air Apportionment Recommendation
 Provide Rationale & Support for Recommendation

The remainder of the quality specification activities were applied to each of the above functions.

2.1.2 Assign Initial Quality Goals

Several factors were taken into account when assigning initial quality goals. Typical command/control quality concerns were considered. SBSDA documentation was studied to identify system quality requirements. Finally, quality requirements surveys were used to obtain additional views about SBSDA quality goals.

Quality Requirements Surveys were sent to the SBSDA acquisition manager and the system development contractor. These surveys identified the major system functions and asked for opinions regarding the

desired level of quality for each function. The following ratings were used to describe the importance of each of the thirteen quality factors.

- | | |
|-----------------|--|
| (E) - EXCELLENT | Great deal of emphasis on the factor; high score is expected; low scores must be corrected. |
| (G) - GOOD | Emphasis on most aspects of the factor; correction of only selected aspects of low scoring areas. |
| (A) - AVERAGE | Maintain an awareness of the factor; incorporate quality considerations where convenient, but with little or no cost impact. |
| (-) - N/A | This factor is either not important or not applicable. |

An example of the survey form is shown in Figure 2.1.2. Appendix C contains copies of the actual forms which were completed by the respondents.

Some problems exist with the Quality Requirement Surveys. First, these surveys are intended to be a means for those responsible for development, operations, and maintenance to indicate the levels of quality which they would like in the final system. It must be noted that these surveys were not completed by the end users, since SBSDA was a proof of concept development and we did not have access to the prospective users. Also, these surveys were completed after SBSDA development was complete and, therefore, probably show some bias toward actual quality in some cases.

The survey results, command/control quality concerns, and system quality requirements were used to arrive at a set of quality goals. These were then modified to account for the effects of complimentary quality factors. Table 2.1.2 shows the resulting initial quality goals for EPAA and RAA.

SOFTWARE QUALITY FACTOR SYSTEM OR SOFTWARE- UNIQUE FUNCTION	PERFORMANCE					DESIGN			ADAPTATION				
	E F F I C I E N C Y	I N T E G R I T Y	R E L I A B I L I T Y	S U R V I V A B I L I T Y	U S A B I L I T Y	C O R R E C T N E S S	M A I N T A I N A B I L I T Y	V E R I F I A B I L I T Y	E X P A N D A B I L I T Y	F L E X I B I L I T Y	I N T E R O P E R A B I L I T Y	P O R T A B I L I T Y	R E U S A B I L I T Y
DEVELOP KNOWLEDGE													
MONITOR ENEMY PERFORMANCE													
ANALYZE ENEMY PERFORMANCE													

NOTE FOR GOAL ENTRIES:

E = EXCELLENT

G = GOOD

A = AVERAGE

BLANK OR N/A =
NOT IMPORTANT OR
NOT APPLICABLE

Figure 2.1.2 Sample Quality Requirements Survey

Table 2.1.2 Resulting Initial Quality Factor Goals

SOFTWARE QUALITY FACTOR SYSTEM OR SOFTWARE UNIQUE FUNCTION	E F F I C I E N C Y	I N T E G R I T Y	R E L I A B I L I T Y	S U R V I V A B I L I T Y	U S A B I L I T Y	C O R R E C T N E S S	M A I N T A I N A B I L I T Y	V E R I F I A B I L I T Y	E X P A N D A B I L I T Y	F L E X I B I L I T Y	I N T E R O P E R A B I L I T Y	P O R T A B I L I T Y	R E U S A B I L I T Y
EPAA													
DEVELOP KNOWLEDGE	A	-	G	-	A	G	E	G	E	G	A	G	E
MONITOR ENEMY PERFORMANCE	G	-	G	-	A	E	E	G	E	G	G	G	E
ANALYZE ENEMY PERFORMANCE	G	-	E	-	E	E	E	E	E	G	G	G	E
RAA													
DEVELOP & ANALYZE AIR APPORTIONMENT RECOMMENDATION	G	A	G	-	E	G	G	G	E	E	G	E	A
PROVIDE RATIONALE & SUPPORT FOR RECOMMENDATION	G	A	G	-	E	G	G	G	E	G	G	E	A

2.1.3 Consider Interrelationships

After arriving at the set of initial quality goals, the technical feasibility of achieving those goals was explored. This involved considering the interrelationships among quality factors. Specifically, the ability to attain high quality for a factor may be adversely affected when it is desired that another factor also have a high level of quality. For example, if software is designed and implemented to be highly expandable, it can be difficult to achieve high levels of EFFICIENCY. Similarly, certain combinations of factors have positive effects, thus reducing the technical difficulty in attaining the desired quality goals. For example, software with high RELIABILITY generally incorporates good ANOMALY MANAGEMENT. This results in improved USABILITY as well.

When the technical feasibility of attaining a certain set of goals was determined to be low, adjustments were made. Table 2.1.3 contains the final quality factor goals, after performing such modifications. Note that the only factor goal that was changed was EFFICIENCY. This was due to the fact that EFFICIENCY is the factor with the most negative interrelationships with other factors. Thus, the overall EFFICIENCY of the system was de-emphasized in order to increase the technical feasibility of achieving the other goals.

Table 2.1.3 Final Quality Factor Goals

SOFTWARE QUALITY FACTOR SYSTEM OR SOFTWARE UNIQUE FUNCTION	E F F I C I E N C Y	I N T E G R I T Y	R E L I A B I L I T Y	S U R V I V A B I L I T Y	U S A B I L I T Y	C O R R E C T N E S S	M A I N T A I N A B I L I T Y	V E R I F I A B I L I T Y	E X P A N D A B I L I T Y	F L E X I B I L I T Y	I N T E R O P E R A B I L I T Y	P O R T A B I L I T Y	R E U S A B I L I T Y
EPAA													
DEVELOP KNOWLEDGE	-	-	G	-	A	G	E	G	E	G	A	G	E
MONITOR ENEMY PERFORMANCE	A	-	G	-	A	E	E	G	E	G	G	G	E
ANALYZE ENEMY PERFORMANCE	A	-	E	-	E	E	E	E	E	G	G	G	E
RAA													
DEVELOP & ANALYZE AIR APPORTIONMENT RECOMMENDATION	A	A	G	-	E	G	G	G	E	E	G	E	A
PROVIDE RATIONALE & SUPPORT FOR RECOMMENDATION	A	A	G	-	E	G	G	G	E	G	G	E	A

2.1.4 Consider Costs

Just as the technical feasibility of attaining factor goals was studied, the guidebook describes a procedure for determining the cost ramifications of attaining the desired level of quality.

Since the SBSDA project had already been completed, we did not apply this portion of the methodology. However, we assessed the procedures for feasibility and considered the validity of the assumptions made by the guidebook. The results are given in section 3.

2.1.5 Specify Criteria

The next step involved specifying the extent that each criterion should contribute to the quality factors. Criteria weighting formulas were developed which define the relative importance of each criterion to each factor. A set of these formulae were produced for each function, since it was desirable to emphasize different qualities for different functions. Table 2.1.5 shows the weighting formulas for the Develop Knowledge function of EPAA.

Table 2.1.5 Criteria Weighting Formulas
For EPAA Develop Knowledge Function

FACTOR		WEIGHTING FORMULA
(EFFICIENCY)	N/A	= (EC)+ (EP)+ (ES)
(INTEGRITY)	N/A	= (SS)
(RELIABILITY)		= .5 (AC)+ .3 (AM)+ .2 (SI)
(SURVIVABILITY)	N/A	= (AM)+ (AU)+ (DI)+ (RE)+ (MO)
(USABILITY)		= .5 (OP)+ .5 (TN)
(CORRECTNESS)		= .4 (CP)+ .4 (CS)+ .2 (TC)
(MAINTAINABILITY)		= .15 (CS)+ .1 (VS)+ .25 (MO)+ .25 (SD)+ .25 (SI)
(VERIFIABILITY)		= .1 (CP)+ .3 (MO)+ .3 (SD)+ .3 (SI)
(EXPANDABILITY)		= .3 (AT)+ .1 (GE)+ 0 (VR)+ .2 (MO)+ .2 (SD)+ .2 (SI)
(FLEXIBILITY)		= .1 (GE)+ .3 (MO)+ .3 (SD)+ .3 (SI)
(INTEROPERABILITY)		= .25 (CL)+ 0 (FO)+ .25 (ID)+ .25 (SY)+ .25 (MO)
(PORTABILITY)		= .5 (ID)+ .25 (MO)+ .25 (SD)
(REUSABILITY)		= .1 (AP)+ .2 (DO)+ .1 (FS)+ .1 (GE)+ .1 (ID)+ .1 (ST)+ .1 (MO)+ .1 (SD)+ .1 (SI)

2.1.6 Specify Metrics

The last procedure in the quality specification process involved specifying the metrics and metric elements which will be used to measure the quality during the system life cycle. This process involves *tailoring out* the metrics which are not considered applicable to the software system. We omitted this step for two reasons. First, we felt that not enough information was available for us to make informed choices. We did not wish to delete potentially useful metrics. Second, we planned to apply *all* metrics to assist us in our evaluation of the framework. In the cases where metrics simply were immeasurable, we planned to mark them "N/A" during the quality evaluation process.

2.1.7 Assess Compliance With Requirements

Guidebook Volume II assumes that the SAM will, at each major software development milestone, compare the measured quality with the specified quality goals. As explained more fully in Paragraph 3.2.4, these comparisons *currently* are of questionable value since no statistical validation has been performed on factor goals and scores. For reference, Figures 2.2.3-1 through 2.2.3-5 display the specified and measured factor scores.

2.1.8 Publish Software Quality Requirements Report

The guidebooks assume that the results of the quality specification process will be documented in a System/Segment Specification. Since that was not an option, a separate report was created. This report, called the Software Quality Requirements Report (SQRR), was published for each decision aid. This report has utility even when the quality requirements themselves become part of a formal specification. The SQRR can and should be used to capture the intermediate results, trade-offs, and rationale which preceded the final quality requirements. This is examined further in section 3.

2.2 SOFTWARE QUALITY EVALUATION

This subsection describes, at a high level, the process of evaluating the software quality of the decision aids. Quality evaluation was performed in accordance with the instructions in the Software Quality Evaluation Guidebook, Volume III.

Each metric was examined to determine if it could be measured. Metrics which were judged as not applicable to the decision aids were not measured. This approach insured that all metrics were considered for validation purposes.

Documentation of the system and software plays an important role during the quality evaluation activities. The guidebooks are intended to be used with software developed in accordance with DoD-STD-2167. Since the SBSDA contractor was not required to use this standard, it was particularly

challenging to apply the worksheets. This certainly affected the time required to take measurements of quality. Also, the resulting quality scores were lower because, in many cases, documentation which was required to answer metric questions did not exist. Nevertheless, the lack of standard documentation had little or no negative effects on our *evaluation of the framework and guidebooks*. This is because our evaluation was performed based on our perceptions of the process, not based on the difficulties with applying the methodology to SBSDA.

2.2.1 Identify Allocation Relationships

The first step in the quality evaluation activity was to determine how quality requirements were allocated from the software functions into CSCIs and units. The condition of the SBSDA documentation made this task particularly difficult. PAR worked closely with us to identify these allocation relationships.

Most large systems are decomposed into two or more CSCIs. This partitioning is then supported by distinct documents for each CSCI, according to 2167. The SBSDA systems, however, were not broken down into multiple CSCIs. Therefore, we were unable to exercise portions of the methodology which pertain to the allocation of software functions into multiple CSCIs. We did, however, consider the effects of using the guidebooks with a multiple CSCI system.

In the case of RAA, several different decompositions into CSCIs and units were found in various places. It was a major exercise to reconcile these into one reasonable allocation.

2.2.2 Apply Worksheets

The next step was to apply the worksheets to the EPAA and RAA development products. Following is a discussion of the activities involved in applying the worksheets.

2.2.2.1 Gather Source Material The worksheets contain questions which must be answered by referencing the appropriate SBSDA documentation. Since the documentation was not structured according to 2167 guidelines, a mapping was needed to show which parts of which documents would be used with each worksheet. PAR assisted us in developing this mapping.

2.2.2.2 Metric Element Evaluation Packages To assist in gathering answers to the metric element questions and information about the worksheets and the framework, a series of forms, collectively called the Metric Element Evaluation Package, was developed. A different package was produced for each worksheet and for each decision aid. Appendix D contains a sample of the forms contained in a package. Each package contained a series of instructions for applying the applicable worksheet. These

instructions described how to document the answers to the metric element questions, and how to capture comments and criticisms about the metric elements, the criteria, and the procedures for measuring them. The instructions also defined the documentation subset which was to be used in answering questions on the worksheet.

Each package contained a Metric Element Scoring Form. This form was used to capture the answers to every worksheet question. In addition, the rationale for the answer and a list of documentation used to arrive at the answer were entered on the form. This form was later used if any discrepancies were found, or if answers to specific metric elements were questioned.

The primary mechanism for collecting information pertaining to the framework was the Metric Element Evaluation Form. This form was completed for every metric element which an evaluator felt had problems. This form captured information about the applicability of the metric element, the phrasing of the question, and the accuracy of the measurement and/or formula. The evaluators were encouraged to enter textual comments expanding on any concerns they had with each metric element.

Finally, every package contained a Worksheet Evaluation Form. This form was used to gather evaluator feedback on the criteria which were measured by the worksheet. Each criterion was examined for applicability to the present life cycle phase and the thoroughness with which this worksheet measured the criterion. As with the other forms, there was a place to enter textual comments, so the evaluators could explain their rationale.

In addition to the instructions and forms, each package contained the complete worksheet for the applicable life cycle phase.

2.2.2.3 Answer Worksheet Questions EPAA was rated by one evaluator and RAA by two. Each evaluator kept track of the time required to research answers to the metric element questions and complete the Metric Element Scoring Forms. Because the SBSDA documentation does not conform to 2167, it was determined that these times were of little value as predictors of effort required to use the guidebooks in the "real world". They do, however, provide a useful means of viewing the relative effort to apply each worksheet. Table 2.2.2 shows the time taken to apply each worksheet.

Table 2.2.2 Time to Apply Worksheets

WORKSHEET	AVG. TIME TO COMPLETE (HRS)	NUMBER OF SAMPLES
0	25	3
1	16	3
2	17	2
3A	15	1
3B	4.5 PER UNIT	6
4A	10	6
4B	8 PER UNIT	6

2.2.3 Score Factors

The last step in quality evaluation which we have performed to date is the scoring of the quality factors. Performing the scoring revealed several problems with the guidebook which are detailed in Section 3.

The scores for RAA functions are shown in Figures 2.2.3-1 and 2.2.3-2. The scores for EPAA functions are shown in Figures 2.2.3-3 through 2.2.3-5.

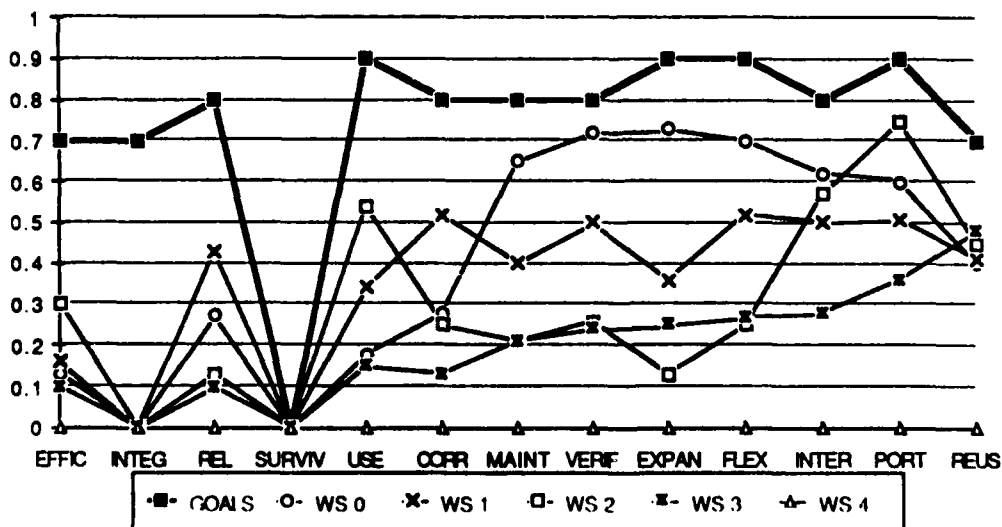


Figure 2.2.3-1 RAA "Develop Apportionment Recommendations"
Factor Goals and Scores

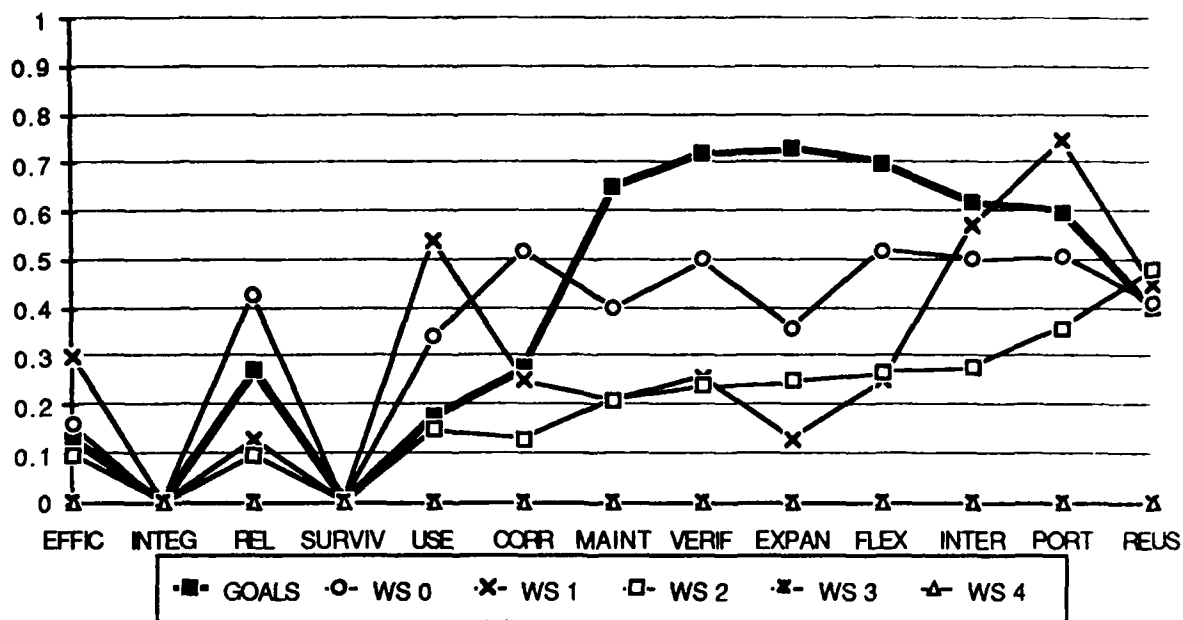


Figure 2.2.3-2 RAA "Provide Rationale for Recommendations"
Factor Goals and Scores

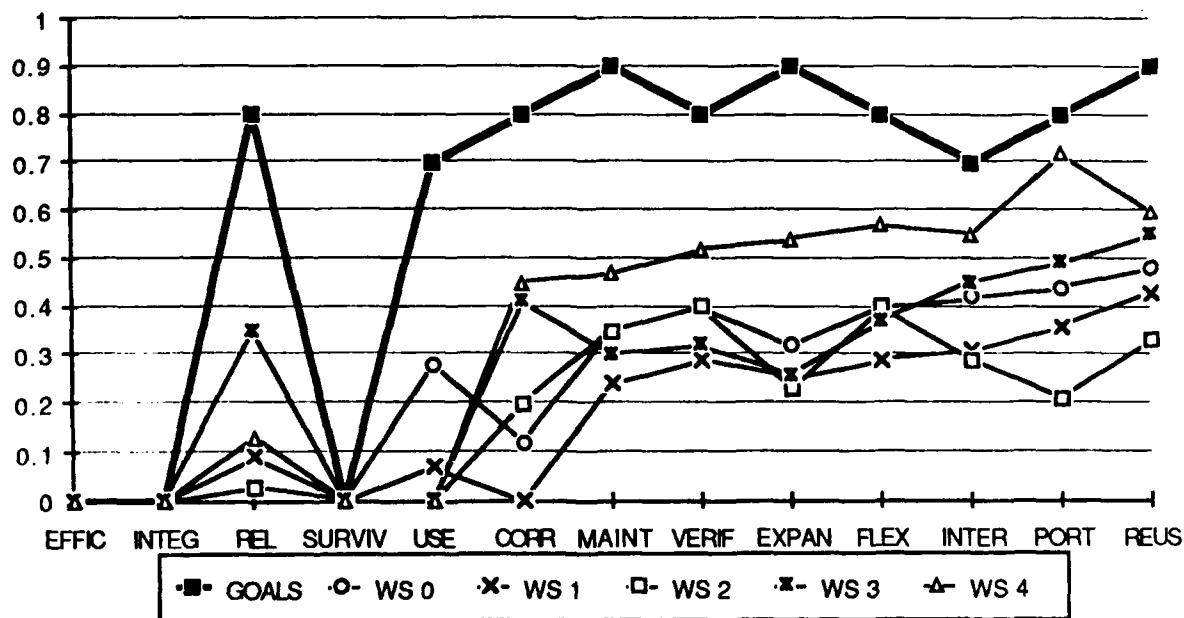


Figure 2.2.3-3 EPAA "Develop Knowledge" Factor Goals and Scores

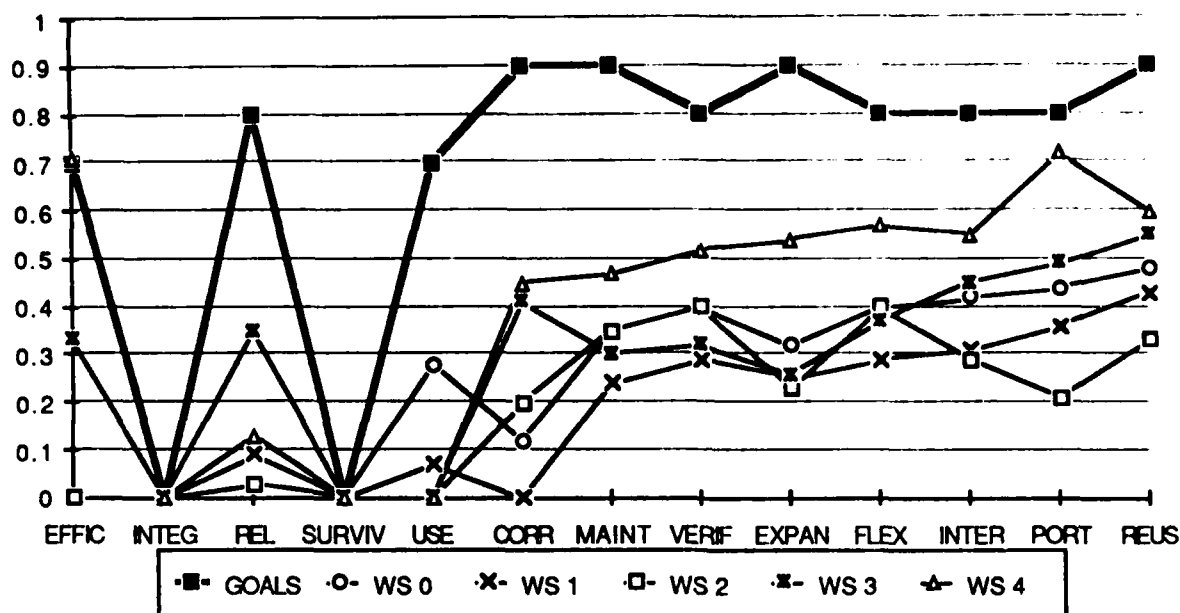


Figure 2.2.3-4 EPAA "Monitor Enemy Performance" Factor Goals and Scores

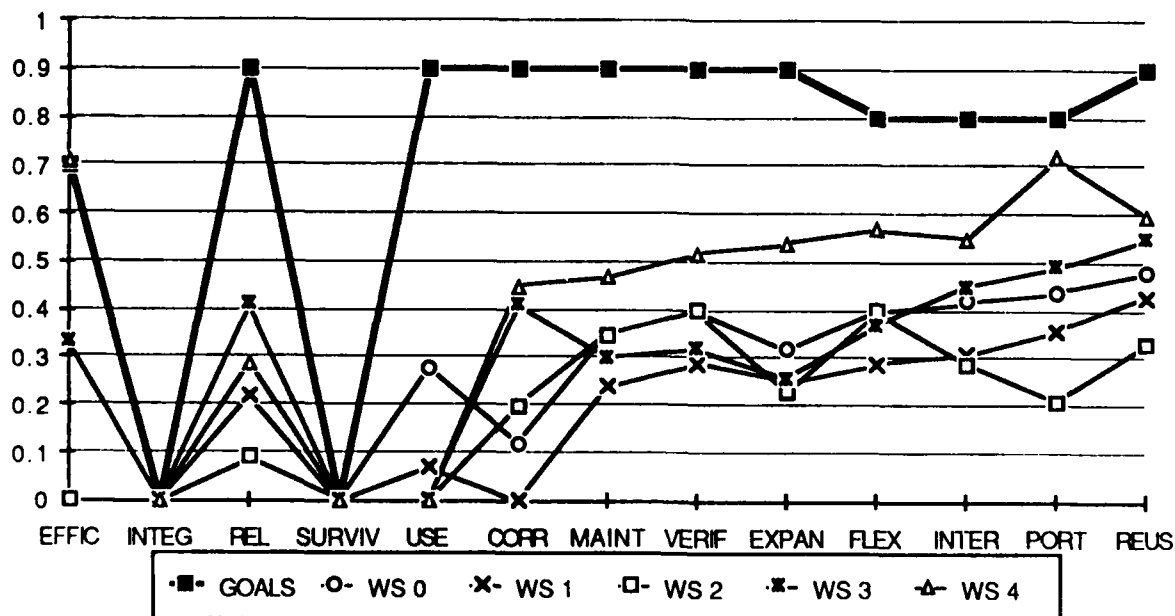


Figure 2.2.3-5 EPAA "Analyze Enemy Performance" Factor Goals and Scores

2.2.4 Analyze Scoring

This step is intended to view factor scores across software development phases and with the specified factor goals. In cases where scoring deficiencies exist, the reasons for those deficiencies are documented. As discussed in Paragraph 3.3.4, criteria and factor scores cannot yet be compared across worksheets or with specified goals. Therefore, until statistical validation has been completed, this step cannot be performed. Aside from the Figures presented in the last paragraph, we performed no analysis of scores.

2.2.5 Recommend Corrective Action

This purpose of this step is to identify and document problems and recommendations based on the analysis of quality factor goals and scores. As previously discussed, we do not believe that meaningful conclusions can be drawn by comparing factor scores with specified goals. Therefore, no problems were identified or recommendations made to address SBSDA quality related issues. Paragraph 3.3.5 discusses an alternative approach to utilize criteria scores in identifying potential problems.

SECTION 3 FINDINGS AND RECOMMENDATIONS

This section presents our major findings and recommendations. It is organized into three main areas. Subsection 3.1 discusses items which relate to the framework itself. Findings in this area involve the factors, criteria, and metrics, but not the procedures for specifying and evaluating quality. Subsections 3.2 and 3.3 contain findings relating to guidebooks 2 and 3, respectively. In particular, they discuss the methodologies of quality specification and quality evaluation as defined by section 4 of each guidebook. Therefore, it is desirable to relate individual findings to the relevant paragraphs in the guidebooks. This has been accomplished by adopting an unusual numbering scheme for these subsections. For each paragraph in these subsections, removing the first two digits (i.e. 3.2 or 3.3) and replacing them with "4" yields the corresponding paragraph number from either guidebook 2 or 3. To further aid in cross-referencing our findings with the related text in the guidebooks, the paragraph headings in this report are the same as the paragraph titles in the guidebook.

3.1 SOFTWARE QUALITY FRAMEWORK

This subsection documents specific findings and recommendations regarding the elements of the software quality framework.

3.1.1 Software Quality Factors

The thirteen software quality factors represent a broad view of acquisition quality concerns. It is important that these factors form a complete set of quality concerns, while avoiding areas which should not be placed in the "quality" category.

The following paragraphs address our findings with regard to the factors. Each factor is defined along with a list of its supporting criteria. Next, it is examined in terms of its relevance to the acquisition process and in the applicability of the criteria which define it. This discussion is not, for the most part, concerned with the adequacy of metrics which define the criteria. This area is treated thoroughly in paragraph 3.1.2.

3.1.1.1 EFFICIENCY

This factor is defined as the relative extent to which system resources are utilized. It is defined by the criteria EFFECTIVENESS-COMMUNICATION, EFFECTIVENESS-PROCESSING, and EFFECTIVENESS-STORAGE.

FINDING:

This factor is not needed if the performance requirements are well stated. If performance requirements are not adequately defined, specifying EFFICIENCY as a quality factor is a poor substitute. It is possible that the potentially difficult job of defining performance requirements will be "hand-waved" or avoided, since this quality requirement may be perceived to provide an alternative. This factor must not be used as a substitute for thorough performance requirements. Further, quality factors are intended to represent *user-oriented* quality concerns. Typically the user is not concerned with how efficient the software is. Rather, the user wants to insure that the *system* (hardware, software and operations) meets his *throughput* requirements. EFFICIENCY does not address these throughput requirements.

RECOMMENDATION:

Remove EFFICIENCY as a quality factor.

This will have little effect on the evaluation of cost and technical feasibility considerations with respect to other factors. Currently, EFFICIENCY is seen to be in conflict with MAINTAINABILITY, VERIFIABILITY, and PORTABILITY. The conflict is based on the assumption that, when efficiency is a concern, the code will be optimized and, therefore be harder to maintain, test, and port. Also more storage and processing resources are needed when coding techniques are used to improve ANOMALY MANAGEMENT, OPERABILITY, MODULARITY, etc. of the software.

However, if EFFICIENCY is eliminated, the *performance and throughput requirements* will drive the system engineering exercise to budget system resources. The result will be that the system may or may not require the "efficient" use of CPU, memory, or I/O resources. Criteria scores will reflect areas where specific steps taken to improve efficiency result in lower scores for other attributes. Developers must account for the impact that this additional code will have on software performance. Ultimately, software characteristics will be considered in the way that CPU and storage resources are budgeted across CSCIs, CSCs and units.

Also, eliminating EFFICIENCY will simplify the quality specification process by reducing the effort required in the evaluation of negative factor interrelationships. Currently, 36 individual negative interrelationships are defined by the guidebook. With the removal of EFFICIENCY, 58% of these relationships will be deleted, leaving only 15 negative relationships to evaluate. Reducing the time needed for the SAM to perform quality specification will aid in the acceptance of the methodology by the SAMs.

Currently, cost considerations of highly efficient software are based primarily on negative interactions

between EFFICIENCY and other factors. The elimination of EFFICIENCY simply means that cost ramifications must reflect the difficulty of budgeting performance requirements. Developers & SAMs need to be aware of the need for resources to solve the problems. Where resources are inadequate, 'efficient' implementation is necessary. This costs more and can result in lower quality for other areas. Thus, trade-offs must be made in cases where system resources do not allow for the use of less efficient coding techniques.

3.1.1.2 INTEGRITY

This factor is defined as the extent to which the software will perform without unauthorized access to code or data. It is defined by the criterion SYSTEM ACCESSIBILITY.

FINDING:

Neither the factor nor the criterion account for the existing DoD standard on secure systems. Developed by the National Computer Security Center, the DoD Trusted Computer System Evaluation Criteria, or DoD-STD-5200.28, provides the basis for testing the effectiveness of security controls, determining the degree to which hardware and software integrity measures make a system suitable for the processing of classified information [13].

RECOMMENDATION:

The factor should be removed from the framework. The integrity required of a software system according to DoD-STD-5200.28 must be established through a careful examination of the data sensitivity and user clearance levels which will be present on the system. This is then used to define the degree of "trust" which must be present in the trusted computer base (TCB) of the system. The standard provides explicit requirements which must be met for five distinct evaluation classes.

The National Computer Security Center conducts an evaluation program in which it works closely with system vendors to certify their systems. In order that an application be considered secure, it must be implemented on a system which has been certified at a level consistent with the application's risk index.

Clearly, since this mechanism addresses both the specification and evaluation of computer security requirements, it is not necessary for the software quality framework to include the INTEGRITY factor. Any time an acquisition manager feels that INTEGRITY is of concern, he should pursue the relevant DoD standard rather than simply specify a quality factor.

3.1.1.3 RELIABILITY

This factor is defined as the extent to which the software will perform without failures during a specified time period. It is defined by the criteria ACCURACY, ANOMALY MANAGEMENT, and SIMPLICITY.

FINDING:

An important attribute of RELIABILITY is lacking. Although a comprehensive test program has a strong positive effect on reliability, the criterion VISIBILITY, which measures the effectiveness of the testing program, is not part of RELIABILITY. Thus, the framework does not detect the negative impact on software reliability when ineffective testing is performed.

The criterion ACCURACY pertains to the software providing the *required* precision in calculations and outputs. As such, this criterion more properly belongs with the factor CORRECTNESS, which has to do with software meeting its requirements. Software which does not meet precision specifications can still be very reliable.

RECOMMENDATION:

Add the criterion VISIBILITY to this factor. Remove the criterion ACCURACY from this factor.

3.1.1.4 SURVIVABILITY

This factor is defined as the extent to which critical software functions will be supported when a portion of the system is inoperable. It is defined by the criteria ANOMALY MANAGEMENT, AUTONOMY, DISTRIBUTEDNESS, RECONFIGURABILITY, and MODULARITY.

FINDING:

The criterion DISTRIBUTEDNESS is defined as those characteristics which determine the degree of geographical or logical separation of software functions within a system. The guidebook suggests that highly distributed systems are more survivable. In fact, the way that the distributed architecture is implemented can either *improve or detract* from overall system survivability. The simple presence of a distributed architecture does not, by itself, contribute to system survivability.

RECOMMENDATION:

Eliminate the criterion DISTRIBUTEDNESS. The factor SURVIVABILITY was added to the framework as a result of a study of Software Quality Measurement for Distributed Systems [5]. In fact, the intent of DISTRIBUTEDNESS, the extent to which software functions are logically or geographically separated, is implicitly measured by the criterion RECONFIGURABILITY. Also,

RECONFIGURABILITY addresses the ability of software to benefit from a distributed architecture, not simply the presence of one.

3.1.1.5 USABILITY

This factor is defined as the effort required to use the software relative to the effort required to implement the software. It is defined by the criteria OPERABILITY and TRAINING.

FINDING:

USABILITY should represent some measure of ease of learning and use of a system *with respect to the complexity of the functions provided*. The existing formula, which relates ease of use to the effort required to develop the software seems off-base.

Important aspects of USABILITY are not currently measured. In particular, XEROX Palo Alto Research Center (PARC) has, over the last 10 years, performed a great deal of research on human engineering issues [14,15]. They found that the usability of a system could be greatly improved by making the user commands orthogonal and mode-less. The results of their research can be seen in the current implementations of the SMALLTALK-80 system, the Apple MACINTOSH [16], and other interfaces which employ high resolution bit mapped displays, overlapping windows, and pull-down or pop-up menus.

The criterion TRAINING is defined as characteristics of software which provide transition from current operation and initial familiarization. A training program is important, but is more properly defined by other system requirements. The adequacy of the training solution cannot be determined without an understanding of the training problem.

RECOMMENDATION:

Reword the formula to relate effort to learn and use the software with either the complexity of functions being performed or the effort which would be required to perform those tasks without software assistance.

Eliminate the criterion TRAINING. In the original framework study [2], the primary emphasis of USABILITY was in ease of use rather than the effort to learn, initially, how to use the software system.

3.1.1.6 CORRECTNESS

This factor is defined as the extent to which the software meets its specifications and conforms to standards. It is defined by the criteria COMPLETENESS, CONSISTENCY, and TRACEABILITY.

FINDING:

The criterion **ACCURACY**, which deals with the software meeting its requirements for precision, is not an attribute of **CORRECTNESS**. Since this factor is concerned with software meeting requirements, **ACCURACY** should be part of the measurement.

The criterion **VISIBILITY** is defined as those characteristics which provide status monitoring of the development and operation. That is, this criterion measures the quality of the testing approach. Since thoroughly testing the software is an important part of insuring its correctness, this criterion should be part of **CORRECTNESS**, but is not.

RECOMMENDATIONS:

Add the criteria **ACCURACY** and **VISIBILITY** to this factor.

3.1.1.7 MAINTAINABILITY

This factor is defined as the effort required to locate and correct software defects. It is defined by the criteria **CONSISTENCY**, **VISIBILITY**, **MODULARITY**, **SELF-DESCRIPTIVENESS**, and **SIMPLICITY**.

This factor is appropriate to the acquisition process, and it is effectively measured by its criteria.

3.1.1.8 VERIFIABILITY

This factor is defined as the effort required to verify that the software meets its specifications, relative to the effort required to implement the software. It is defined by the criteria **VISIBILITY**, **MODULARITY**, **SELF-DESCRIPTIVENESS**, and **SIMPLICITY**.

FINDING:

As measured by worksheets 0, 1, and 2, this factor is not differentiated from the factor **FLEXIBILITY**. Since very flexible software can actually be more difficult to verify, these factors should be distinguished on all worksheets.

The formula defining this factor is stated in terms of effort to develop the software versus effort to verify the software. Rather than "grade" the developer on how difficult or time-consuming it is to verify the software, the factor should represent how thoroughly the software will be verified. This, in turn, is partly a function of how verifiable each requirement is.

RECOMMENDATION:

The criterion **VISIBILITY** differentiates the two factors **VERIFIABILITY** and **FLEXIBILITY**. Therefore,

these should be measured on worksheets 0, 1, and 2. DoD-STD-2167 requires that the Software Development Plan and System Segment Specification be included in the functional baseline. As such, these documents are available for scoring with worksheet 0. Both documents address the testing program, therefore VISIBILITY can be measured as early as worksheet 0.

Change the factor rating formula to stress the verifiability of the requirements and the resulting thoroughness of verification.

3.1.1.9 EXPANDABILITY

This factor is defined as the effort required to increase software capabilities or performance relative to the effort required to implement the software. It is defined by the criteria AUGMENTABILITY, GENERALITY, VIRTUALITY, MODULARITY, SELF-DESCRIPTIVENESS, and SIMPLICITY.

FINDING:

Requirements to expand the system should be specified in terms of functional and performance requirements whenever possible. The probability of successfully augmenting software capabilities is much greater if this requirement is anticipated and planned for, than if it is simply included as a quality requirement.

RECOMMENDATION:

Remove EXPANDABILITY as a quality factor. If this step is too radical, add a warning to the effect that this factor must not be used as a substitute for thorough functional and performance requirements.

3.1.1.10 FLEXIBILITY

This factor is defined as the effort required to change the software to meet different requirements. It is defined by the criteria GENERALITY, MODULARITY, SELF-DESCRIPTIVENESS, and SIMPLICITY.

FINDING:

As measured by worksheets 0, 1, and 2, this factor is not differentiated from the factor VERIFIABILITY. Since very flexible software can actually be more difficult to verify, these factors should be distinguished on all worksheets.

RECOMMENDATION:

The criterion GENERALITY differentiates these two factors. Therefore, it should be measured on worksheets 0, 1, and 2.

3.1.1.11 INTEROPERABILITY

This factor is defined as the effort required to couple the software to the software of another system, relative to the effort required to implement the original software. It is defined by the criteria COMMONALITY, FUNCTIONAL OVERLAP, INDEPENDENCE, SYSTEM COMPATIBILITY, and MODULARITY.

FINDING:

This factor applies only if it is expected that the system will interface with other systems at some point in the future, but the *characteristics of those systems are not known now*. This is confirmed by the fact that INTEROPERABILITY is placed in the group of factors having to do with system adaptation rather than with the performance or design factors.

If the specific characteristics of the interoperating systems are known, they will be reflected in interface specifications. Since INTEROPERABILITY applies to interfaces with as yet unknown systems, the characteristics of the interfaces are not yet known. The criteria COMMONALITY, FUNCTIONAL OVERLAP, and SYSTEM COMPATIBILITY require that the characteristics of the interoperating system(s) be known during requirements definition and software design. Therefore, these criteria are unnecessary.

Confusion existed regarding this factor even in the original framework study [2]. The criteria DATA COMMONALITY and COMMUNICATIONS COMMONALITY were used to measure the degree to which interface standards and protocols had been established and followed. This implies that requirements exist to interface with some known system. However, the effect of poor INTEROPERABILITY, according to the study, was not seen as important until the transition phase of the life cycle. This implies that requirements to interface with another system were added after the system was operational.

The remaining criteria, INDEPENDENCE and MODULARITY, do not adequately measure the relative effort of coupling the software with some as-yet unknown system. In fact, it is unlikely that any software attributes could adequately characterize this quality.

RECOMMENDATION:

Delete this factor. If this is too extreme, then the definition of this factor must be clarified. Remove the criteria COMMONALITY, FUNCTIONAL OVERLAP, and SYSTEM COMPATIBILITY from the definition of this factor. Explore additional criteria which might better characterize the effort to couple the system with another, sometime in the future.

3.1.1.12 PORTABILITY

This factor is defined as the effort to transport the software to another environment, relative to the effort required to implement the original software. It is defined by the criteria INDEPENDENCE, MODULARITY, and SELF-DESCRIPTIVENESS.

FINDING:

The criteria APPLICATION INDEPENDENCE and INDEPENDENCE are both relevant to this factor, but only INDEPENDENCE is currently used in its definition. It is not clear why APPLICATION INDEPENDENCE is not also used to measure this factor.

RECOMMENDATION:

It is suggested that the criteria APPLICATION INDEPENDENCE and INDEPENDENCE be combined (see paragraph 3.1.2.3).

3.1.1.13 REUSABILITY

This factor is defined as the effort required to reuse a component of the software in another application, relative to the effort required to initially implement the software. It is defined by the criteria APPLICATION INDEPENDENCE, DOCUMENT ACCESSIBILITY, FUNCTIONAL SCOPE, GENERALITY, INDEPENDENCE, SYSTEM CLARITY, MODULARITY, SELF-DESCRIPTIVENESS, and SIMPLICITY.

FINDING:

The criteria APPLICATION INDEPENDENCE and INDEPENDENCE both apply to this factor, but these criteria measure very similar attributes. Two criteria are not necessary.

RECOMMENDATION:

It is suggested that the criteria APPLICATION INDEPENDENCE and INDEPENDENCE be combined (see paragraph 3.1.2.3).

3.1.1.14 Other Candidate Factors With exceptions noted above, the framework is quite complete in describing acquisition quality concerns. Only one additional factor is suggested, and even this one could be seen to be an extension of RELIABILITY. We believe, however, that this factor is different enough from RELIABILITY that it should stand alone.

3.1.1.14.1 TRUSTWORTHINESS

RELIABILITY is concerned with the likelihood of a failure - any failure. No consideration is given to the consequences of the failure. D.L. Parnas, among others, has proposed that highly reliable software might not be trustworthy [17]. This would be the case when a failure, even if very unlikely, could be catastrophic. Software is said to be trustworthy when the probability of a catastrophic flaw is acceptably low.

TRUSTWORTHINESS is particularly important in "high reliability" applications when human lives are affected. Clearly, many C³ systems have the potential for catastrophic failure. Such systems are candidates for TRUSTWORTHINESS measures rather than simple RELIABILITY measures.

This is an area with high potential payoffs. The state of the practice in software engineering is such that TRUSTWORTHINESS is very difficult and expensive to verify with conventional testing techniques - if it can be verified at all. Therefore, if adequate criteria and metrics can be defined for use in the early life cycle phases, it will be possible to more cost effectively improve software TRUSTWORTHINESS.

3.1.2 Criteria and Metrics

The thirteen quality factors are defined by twenty nine quality criteria. These are software oriented attributes which represent quality. Just as factors are defined largely by the criteria which comprise them, each criterion, in turn, is defined by a set of software metrics and metric elements. Over three hundred metric elements are currently defined. These consist of questions on the worksheets, each of which measures some aspect of the software development products. These questions should be clearly stated, with a minimum of subjectivity. It is important that the criteria are clearly and completely defined by the metrics. Our findings address the way in which the criteria are defined by their metrics. Figure 3.1.2 presents the relationships among factors and criteria in the revised framework.

CRITERIA \ FACTORS	RELIABILITY	SURVIVABILITY	USABILITY	TRUSTWORTHINESS	CORRECTNESS	MAINTAINABILITY	VERIFIABILITY	FLEXIBILITY	PORTABILITY	REUSABILITY
ACCURACY	*	*	*	*	*	*	*	*	*	*
ANOMALY MANAGEMENT	*	*	*	*		*	*	*	*	*
AUTONOMY		*								
COMPLETENESS	*	*	*	*	*	*	*	*	*	*
CONSISTENCY	*			*	*	*	*	*	*	*
DOCUMENT ACCESSIBILITY										*
FUNCTIONAL SCOPE										*
GENERALITY								*		*
INDEPENDENCE									*	*
MODULARITY	*	*	*	*		*	*	*	*	*
OPERABILITY			*							
RECONFIGURABILITY		*								
SELF-DESCRIPTIVENESS	*	*	*	*		*	*	*	*	*
SIMPLICITY	*	*	*	*		*	*	*	*	*
SYSTEM CLARITY										*
TRACEABILITY	*	*	*	*	*	*	*	*	*	*
VISIBILITY	*	*	*	*	*	*	*	*	*	*

Figure 3.1.2 Revised Framework

3.1.2.1 ACCURACY

This criterion is concerned with the precision of calculations and their results.

FINDING:

The concepts of accuracy and precision are related but have distinctly different meanings, although they are used interchangeably in common English. In an engineering sense, accuracy has to do with the amount of error in a result. Precision involves the amount of granularity with which the result is represented. Thus, a highly precise result can be inaccurate and vice versa. Although the name of this criterion is ACCURACY, the definition uses the word "precision". The metric elements used to measure this criterion measure both accuracy and precision.

RECOMMENDATION:

Revise the definition of this criterion to more "accurately" reflect the contributions of both precision and accuracy. Split the metric ACCURACY CHECKLIST to create a new metric to measure precision.

Measures of accuracy and precision should be distinguished. Accordingly, a new metric, PRECISION CHECKLIST, should be created from those metric elements in ACCURACY CHECKLIST which actually measure precision.

Add metrics to worksheet 3A, 3B, 4A, and 4B for measuring AC.1, since the design has a strong effect upon both accuracy and precision.

3.1.2.2 ANOMALY MANAGEMENT

This criterion is concerned with continuity of operations and recovery from exceptional conditions.

FINDING:

Transaction loss and database corruption are important anomalies which are not addressed. In transaction processing systems, a failure might occur after a disk file has been updated but before the updated index can be written. In such a case, the transaction must be rolled back to eliminate the inconsistency.

RECOMMENDATION:

Add new metrics to measure the use of transaction recovery mechanisms. Metrics should also be used to measure the use of file analyzers which can identify, and perhaps correct, inconsistent conditions. A new metric, TRANSACTION RECOVERY, should be measured on all worksheets. A new metric, DATABASE INTEGRITY, should be measured on all worksheets.

3.1.2.3 APPLICATION INDEPENDENCE

This criterion is concerned with the characteristics of independence of the software from specific DBMS, microcode, computer architecture, and algorithms.

FINDING:

The metrics do not account for several important characteristics which relate to the independence of the application software. The following are important software characteristics which contribute strongly to software independence:

- the use of a high order language
- use of a fourth generation language
- use of a portable operating system, e.g. UNIX
- use of a DBMS which is implemented on a variety of machines, e.g. ADABAS, ORACLE

Also, only very fine distinctions exist between this criterion and INDEPENDENCE. Both criteria should apply equally to the factors they support, i.e. INTEROPERABILITY, PORTABILITY, and REUSABILITY.

RECOMMENDATION:

Add metrics to measure the use of the above characteristics. Also, combine this criterion with INDEPENDENCE.

A new metric, IMPLEMENTATION LANGUAGE, should measure characteristics such as the portability of the chosen language and the extent to which machine-specific language extensions have been used. In addition, this metric should measure the extent to which fourth generation languages have been used. This metric should be measured on all worksheets, since both standards and actual use can be measured.

A new metric, OPERATING SYSTEM PORTABILITY, should measure the extent to which the operating system used is portable across numerous computer systems, e.g. UNIX. It should also measure the extent to which machine-specific features of such an operating system have been used. This metric should be measured on all worksheets, since both standards and actual use can be measured.

A new metric, DBMS PORTABILITY, should measure the extent to which the DBMS used, if any, is supported across a variety of machines, e.g. ADABAS, ORACLE. This metric should be measured on all worksheets, since both standards and actual use can be measured.

3.1.2.4 AUGMENTABILITY

The criterion AUGMENTABILITY is defined as those characteristics which provide for expansion of capability for functions and data.

FINDING:

Both the criterion definition and the associated metrics are somewhat ambiguous. It is not clear whether they measure the utilization of resources *as the system is currently configured*, or based on the maximum *architectural limits* of the system. In any case, it is felt that the ability to expand the functions and/or data of the software depends on many more characteristics than memory, CPU, or mass storage limits. If a requirement exists for the system to be expanded, then this requirement should be specified in terms of increased capacity, performance, etc. If this is done, the need for this criterion and the associated factor EXPANDABILITY goes away.

RECOMMENDATIONS:

Remove this criterion, along with the factor EXPANDABILITY (see paragraph 3.1.1.9).

3.1.2.5 AUTONOMY

This criterion is concerned with the software non-dependency on interfaces and functions.

FINDING:

Clearly, the SURVIVABILITY of a software system is affected to the extent that the system is dependent on the correct functioning of external systems. However, this criterion measures the AUTONOMY of software in inappropriate ways. It is not so important how much an interface is used or how much code supports the interface, but rather how well the remaining software is isolated from this interface. A good way to accomplish this isolation is through layering, as in the 7 layer standard for Open System Interconnection. The metric which asks if each CPU has a separate power source may be relevant to AUTONOMY, but is not a software characteristic and does not belong in the framework.

RECOMMENDATIONS:

Metrics for this criterion should be changed considerably. Remove AU.2(1) from worksheet 0, which asks about CPU power sources. Change other metrics to focus on the partitioning of the interface software and the extent to which the remainder of the application depends on a proper interface(s).

Add metrics to worksheets 3A, 3B, 4A, and 4B, since measurements taken from the design and code will be much more indicative of real AUTONOMY than measurements taken from the functional description or preliminary design.

3.1.2.6 COMMONALITY

This criterion is concerned with the use of interface standards which address protocols, data representations, and routines.

FINDING:

In measuring the COMMONALITY which the software will have with other, interoperating systems, this criterion misses the point. The more interfaces exist and the more these interfaces are used, the lower the score will be. The metrics penalize for having and using interfaces, regardless of how common the protocols and data formats are.

In any case, this criterion is only used with the factor INTEROPERABILITY, which we have recommended be deleted. As mentioned previously, if concrete requirements exist to interface with another system, then requirements and interface specifications must exist to support these interfaces, in which case this criterion is not needed.

RECOMMENDATIONS:

Remove this criterion from the framework, along with the factor INTEROPERABILITY (see paragraph 3.1.1.11).

3.1.2.7 COMPLETENESS

This criterion is concerned with those characteristics which indicate the required functions have been fully implemented.

FINDING:

COMPLETENESS is measured on worksheet 0, in part, by the number of problem reports to date. The System/Software Requirements Analysis phase is too early to count software problem reports, which are typically not produced during requirements analysis.

RECOMMENDATION:

Delete the metric element CP.1(11) from this criterion on worksheet 0.

3.1.2.8 CONSISTENCY

This criterion measures the extent to which uniform design techniques and notation are used.

FINDING:

This criterion was well measured on each worksheet.

3.1.2.9 DISTRIBUTEDNESS

This criterion is concerned with the degree to which software functions are geographically or logically separated in the system.

FINDING:

As discussed in paragraph 3.1.1.4, the degree of DISTRIBUTEDNESS does not necessarily relate to the SURVIVABILITY of a system.

RECOMMENDATIONS:

Per the earlier recommendation, eliminate this criterion.

3.1.2.10 DOCUMENT ACCESSIBILITY

This criterion measures characteristics which provide for easy access to software and selective use of its components.

FINDING:

This criterion was well measured on each worksheet.

3.1.2.11 EFFECTIVENESS-COMMUNICATIONS

This criterion is concerned with the usage of communications resources in performing functions.

FINDING:

This criterion supports only the factor EFFICIENCY, and we have recommended that EFFICIENCY be deleted (see paragraph 3.1.1.1).

RECOMMENDATIONS:

Delete this criterion.

3.1.2.12 EFFECTIVENESS-PROCESSING

This criterion is concerned with the usage of processing resources in performing functions.

FINDING:

This criterion supports only the factor EFFICIENCY, and we have recommended that EFFICIENCY be deleted (see paragraph 3.1.1.1).

RECOMMENDATIONS:

Delete this criterion.

3.1.2.13 EFFECTIVENESS-STORAGE

This criterion is concerned with the usage of storage resources in performing functions.

FINDING:

This criterion supports only the factor EFFICIENCY, and we have recommended that EFFICIENCY be deleted (see paragraph 3.1.1.1).

RECOMMENDATIONS:

Delete this criterion.

3.1.2.14 FUNCTIONAL OVERLAP

This criterion measures characteristics which provide common functions to more than one interoperating system.

FINDING:

This criterion supports only the factor INTEROPERABILITY, and we have recommended that INTEROPERABILITY be deleted (see paragraph 3.1.1.11).

RECOMMENDATIONS:

Delete this criterion.

3.1.2.15 FUNCTIONAL SCOPE

This criterion is concerned with characteristics which provide commonality of functions among applications.

FINDING:

Metrics for this criterion fail to measure the use of some design techniques which would contribute to high FUNCTIONAL SCOPE scores. Examples are the use of a general query language rather than rigid, fixed field queries and the use of a report generator rather than fixed report formats.

Metric elements used on worksheets 3B and 4B are not adequate. The single metric element used on 3B asks if the unit performs a single function - a question which is also (appropriately) used with modularity. Worksheet 4B adds another metric element which asks if a description of the unit's

functions is in the comments. This attribute is covered in the criterion SELF-DESCRIPTIVENESS.

RECOMMENDATIONS:

Add metrics to measure the use of general report formatting and query facilities.

Expand the metric elements on worksheets 3 and 4 to more thoroughly measure attributes which are unique to this criterion.

3.1.2.16 GENERALITY

This criterion measures characteristics which provide breadth to the functions performed with respect to the application.

FINDING:

This criterion is not measured by worksheets 0, 1, or 2. If GENERALITY is considered desirable, then the earlier worksheets should measure attributes which can improve the GENERALITY of the software.

The metric element GE.1(1) on worksheet 4A awards a better score to units which are called by a large number of other units. This is a poor measure of GENERALITY, since a unit which is called by only one other unit might be very general, while one which is called by many other units might still be quite application specific.

The metric element GE.2(4) on worksheet 4A awards a better score to units which are free from range tests or reasonable tests. This might simply be an indication of a unit which does not protect itself well, not necessarily an indication of GENERALITY.

The use of tools which can improve GENERALITY, such as data dictionaries is not measured.

RECOMMENDATIONS:

Worksheets 0, 1, and 2 should be modified to include measures of GENERALITY.

Metrics should be added to account for the use of tools which can yield more general systems.

Delete metric elements GE.1(1) and GE.2(4) from worksheet 4A.

3.1.2.17 INDEPENDENCE

This criterion is concerned with characteristics which determine the non-dependency of the software on the computing system, operating system, utilities, I/O routines and libraries.

FINDING:

Important characteristics of independence are not measured. For example, the choice of database manager and communications protocols can influence INDEPENDENCE. Other items which can strongly influence this attribute are the isolation of software to perform interprocess communication, task synchronization, and input/output.

Also, no clear distinction can be made between this criterion and APPLICATION INDEPENDENCE.

RECOMMENDATION:

Add metrics to measure additional attributes which affect INDEPENDENCE. Combine this criterion with APPLICATION INDEPENDENCE.

3.1.2.18 MODULARITY

This criterion is concerned with characteristics which provide well structured, highly cohesive, minimally coupled software.

FINDING:

It measures the characteristics of coupling and cohesion on worksheets 0 through 4. These concepts only have relevance when used to characterize a physical design. They should not be applied to higher levels of abstraction, i.e. functional descriptions of the software.

RECOMMENDATION:

Do not measure coupling and cohesion until the preliminary design phase, worksheet 2.

3.1.2.19 OPERABILITY

This criterion is concerned with the usability of inputs and outputs to the software, as well as procedures for operating the software.

FINDING:

The metrics on worksheets 3A and 4A for this criterion addresses the reporting of errors. Several important characteristics of OPERABILITY are ignored.

RECOMMENDATIONS:

Add metrics to worksheets 3A and 4A to measure such user interface characteristics as display clarity, data input procedures, menu presentation, command lines, pointing device, blinking, highlighting, color, response time, and the use of indicator lights and audible alerts.

3.1.2.20 RECONFIGURABILITY

This criterion is concerned with characteristics which provide for continuity of operations when one or more processors, storage units, or communication links fail.

FINDING:

This criterion can be strongly influenced by the use of fault tolerant architectures. No measurements are made to determine the extent of fault tolerance in the system or software design.

Metric elements on worksheet 2 are system-level questions and don't get down to the CSCI level of design, as would be appropriate for this worksheet.

This criterion is not measured at all on worksheets 3 or 4.

RECOMMENDATION:

Add metrics to this criterion which characterize the design of fault tolerance.

Change the character of the metric elements on worksheet 2 to focus more on the CSCI being measured rather than the overall system.

Since RECONFIGURABILITY is a key concept to SURVIVABILITY, this criterion should be measured during the detailed design and coding phases. Add metric elements to characterize the degree to which the design and implementation support RECONFIGURABILITY.

3.1.2.21 SELF-DESCRIPTIVENESS

This criterion measures the characteristics which provide an explanation of the implementation of functions.

FINDING:

The metric elements for this criterion address, on worksheets 0 and 1, the establishment of standards aimed at commenting code. The SELF-DESCRIPTIVENESS of the detailed design is not measured.

Worksheet 3B contains only one metric element which asks, essentially, if the unit adheres to the standard. This single metric element is not adequate for the detailed design phase.

RECOMMENDATIONS:

Add metric elements to worksheets 0 and 1 to address standards for design representation, e.g. PDL.

Add metric elements to worksheet 3B which measure the SELF-DESCRIPTIVENESS of the design in a similar manner to the way that metric elements on worksheet 4B measure the code.

3.1.2.22 SIMPLICITY

This criterion is concerned with characteristics which provide for software implementation in the most noncomplex and understandable manner.

FINDING:

This criterion seemed well measured.

3.1.2.23 SYSTEM ACCESSIBILITY

This criterion is concerned with the control and audit of access to the software and data.

FINDING:

As discussed in paragraph 3.1.1.2, these characteristics are comprehensively treated in DoD-STD-5200.28, the DoD Trusted Computer System Evaluation Criteria [13]. The metrics which define this criterion do not adequately reflect this government standard.

RECOMMENDATION:

This criterion should be removed from the framework, along with the factor INTEGRITY.

3.1.2.24 SYSTEM CLARITY

This criterion is concerned with characteristics which provide for a clear description of program structure.

FINDING:

Several important characteristics are not currently measured. Worksheets 0, 1, and 2 are lacking in metrics for this criterion. Metric elements on worksheet 0 ask if I/O has been isolated from computation. This is generally too early to answer such detailed questions.

On worksheet 3B, the metric elements give low scores to units which reference data items, call other units, or use DO/FOR loops. It would seem that all these constructs can be used in a clear manner - probably much more clear than an alternative method which does not use them.

Metric elements ST.1(1) and ST.1(4) on worksheet 3B are redundant.

Metric elements on worksheet 3B ask a number of questions about interfaces and data items, but the definition of these terms is not clear. By interface, the metric elements might mean the input and output parameters used to call this unit plus the input and output parameters used when this unit calls subordinate units plus all data exchanges with *all* other units through parameters, global data structures, and files. The meaning of "data items" is unclear, particularly in the case of data structures. Is an array to be considered as one item or many? Clearly, the scores depends heavily upon one's interpretation of these terms.

RECOMMENDATION:

Add metrics to measure the use of structured programming practices, consistent calling conventions, and consistent error processing procedures. Additional metrics should be added to worksheets 0, 1, and 2 to support early detection of potential SYSTEM CLARITY problems.

Remove metric elements ST.1(4), ST.2(2), ST.2(3), and ST.2(4) from worksheet 3B.

Clarify the use of the terms "interfaces" and "data items".

3.1.2.25 SYSTEM COMPATIBILITY

This criterion measures characteristics which provide for hardware, software, and communications compatibility of two systems.

FINDING:

This criterion supports only the factor INTEROPERABILITY, and we have recommended that INTEROPERABILITY be deleted (see paragraph 3.1.1.11).

RECOMMENDATIONS:

Delete this criterion.

3.1.2.26 TRACEABILITY

This criterion concerns the characteristics of showing the allocation of requirements across several levels

of documentation, to the actual implementation (code).

FINDING:

Currently, TRACEABILITY is not measured on worksheet 0. Since 2167 requires that TRACEABILITY be shown from the System/Segment Specification to the Software Requirements Specification, TRACEABILITY should be measured on worksheet 0.

The metric elements do not measure both dimensions of TRACEABILITY. They should ask if:

- a. all high level requirements have been allocated down to detailed requirements
- b. all detailed requirements trace back to high level requirements.

This is important in order to insure that all requirements are being addressed, and to flag any requirements which "creep in" without actually satisfying system level requirements.

RECOMMENDATION:

Add metrics which measure TRACEABILITY to worksheet 0. Add complimentary metric elements to measure both dimensions of TRACEABILITY.

3.1.2.27 TRAINING

This criterion is concerned with characteristics which provide a transition from current operation and provide initial familiarization.

FINDING:

As discussed in paragraph 3.1.1.5, this criterion attempts to measure the effectiveness of a training program which should be thoroughly addressed elsewhere in the system requirements. Further, TRAINING should not be a focus of USABILITY. It is important how readily the software can be learned, which is largely a function of the complexity of software functions and the clarity of the user interface. These attributes are measured elsewhere. TRAINING is not necessary as a distinct criterion.

RECOMMENDATIONS:

Delete this criterion.

3.1.2.28 VIRTUALITY

This criterion measures characteristics which allow the user to remain ignorant of the physical, logical, or topological details of the system.

FINDING:

This criterion supports only the factor EXPANDABILITY, and we have recommended that EXPANDABILITY be deleted (see paragraph 3.1.1.9).

RECOMMENDATIONS:

Delete this criterion.

3.1.2.29 VISIBILITY

This criterion concerns characteristics which provide status monitoring of the development, especially the testing program.

FINDING:

Visibility is not currently measured on worksheets 0, 1, or 2. Since DoD-STD-2167 requires that the testing philosophy be addressed early in the development - in the System/Segment Specification - this criterion should be measured early as well.

By the code and unit testing phase, the testing program should be fairly mature. Worksheets 4A and 4B place far too little emphasis on the specifics of the testing program this far into the development process.

RECOMMENDATION:

Add metrics to measure visibility on worksheets 0, 1, and 2. Expand the metrics on worksheets 4A and 4B.

3.2 SOFTWARE QUALITY SPECIFICATION

This subsection identifies specific findings and recommendations which relate to guidebook, Volume II - Specification of Software Quality Attributes - Software Quality Specification Guidebook. In particular, this subsection focuses on the methodology described by section 4 of the guidebook.

3.2.0 Software Quality Specification Methodology

FINDING:

When applying the methodology outside the life cycle context, the results of quality specification cannot be incorporated into the System/Segment Specification (SSS). Even when quality requirements are documented in the SSS, many of the inputs to the specification process and rationale for choices made are not captured. A mechanism is needed to document the quality specification process, along with the intermediate results and final quality goals.

RECOMMENDATION:

Add a report definition and instructions to guidebook II for the purpose of capturing the above information. This report might be called the Software Quality Requirements Report (SQRR). A proposed table of contents, which was used for RAA, is contained in Appendix E.

3.2.1 Select and Specify Quality Factors

FINDING:

This paragraph is a good introduction to the following subparagraphs.

3.2.1.1 Identify Functions (Step 1)

FINDING:

This paragraph describes the process of identifying the major system functions which are supported by software. The procedures that are described do not consider the fact that the system has likely been functionally decomposed already. In order to reduce the effort of specifying and evaluating software quality, it is important that quality requirements be developed for the same functional decomposition used elsewhere in the specifications.

RECOMMENDATION:

This paragraph should stress the importance of using the same set of "major system functions" that are used in the functional specification.

3.2.1.2 Assign Quality Factors and Goals (Step 2)

FINDING:

This paragraph describes several activities which should be performed to assign initial quality factor goals. Throughout this discussion, the importance of each factor is considered in terms of "excellent", "good", "average", and "N/A". This terminology is extremely subjective, and can mean different things to different people.

This problem is compounded by the definitions of the thirteen quality factors. It is unlikely that the person(s) following the procedures in the guidebook have a complete understanding of every quality factor, or the ramifications of the various goal levels. Further, even two people who are well-versed in the software quality framework and the methodology described in the guidebooks might not agree on the distinctions between goal levels for any given factor. For example, what is the difference between a system with excellent SURVIVABILITY .vs. good SURVIVABILITY?

The process of establishing factor goals is too subjective and too dependent on terminology (factor and criteria definitions) which can easily be misinterpreted.

RECOMMENDATION:

A less subjective method must be found to determine software quality goals. This procedure should have greater repeatability than the present one. The proposed method would base factor goals on specific quality related items, much like quality measurements are based on specific software metrics. In fact, "acquisition metrics" could be developed which define the quality factors from a user's perspective, in the same way that the existing criteria and metrics define the factors from a software perspective (see Figure 3.2.1.2-1).

With this acquisition framework, the users and others involved in the software acquisition could answer specific questions (metric elements). The answers would be used to compute criteria and factor goals. These goals would then be examined for technical and cost feasibility, just as "Excellent", "Good", and "Average" are. In order to be useful, these numeric goals must be normalized so they can be meaningfully compared with measured factor scores.

This procedure would be much more objective than the current methodology, and would require little or *no learning curve on the part of users who must respond to the quality requirements survey.* An example of acquisition criteria and metrics is shown in Figure 3.2.1.2-2. In this figure, three criteria are proposed for the factor PORTABILITY. One of the acquisition criteria is further defined by three metrics.

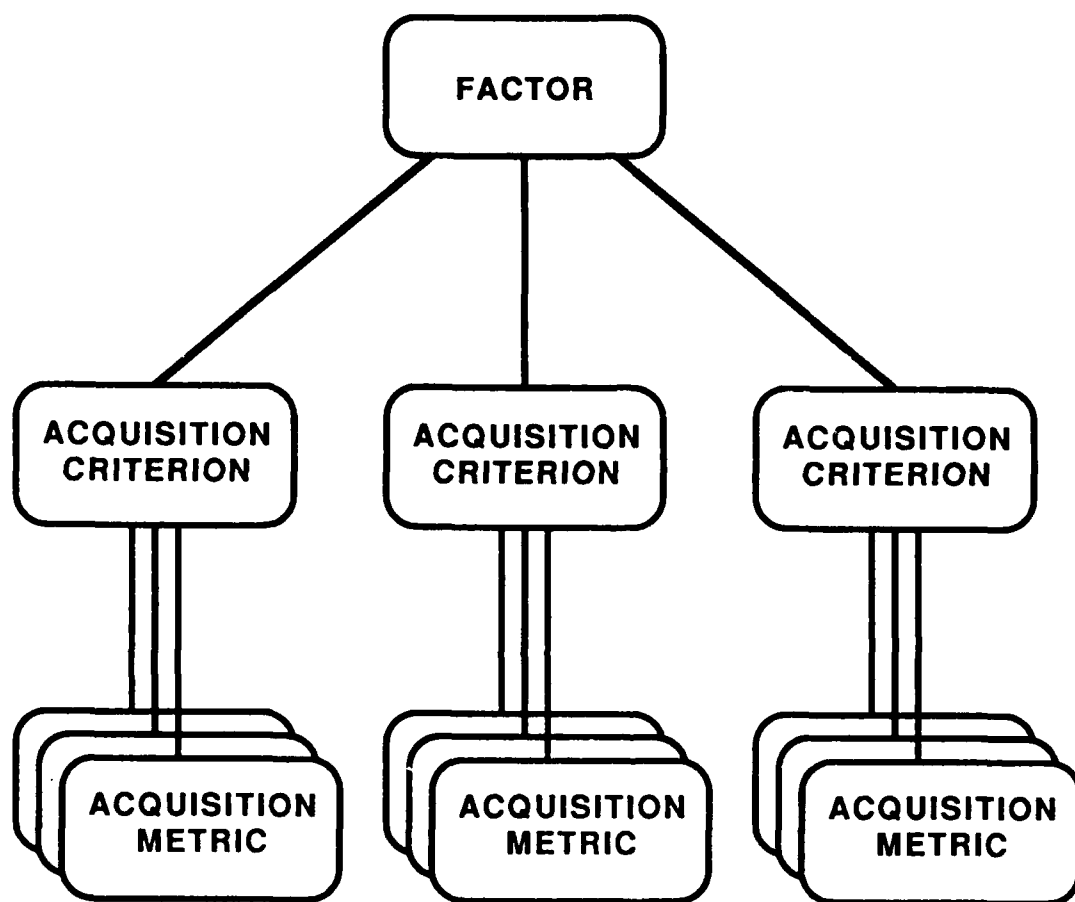


Figure 3.2.1.2-1 Acquisition Hierarchy

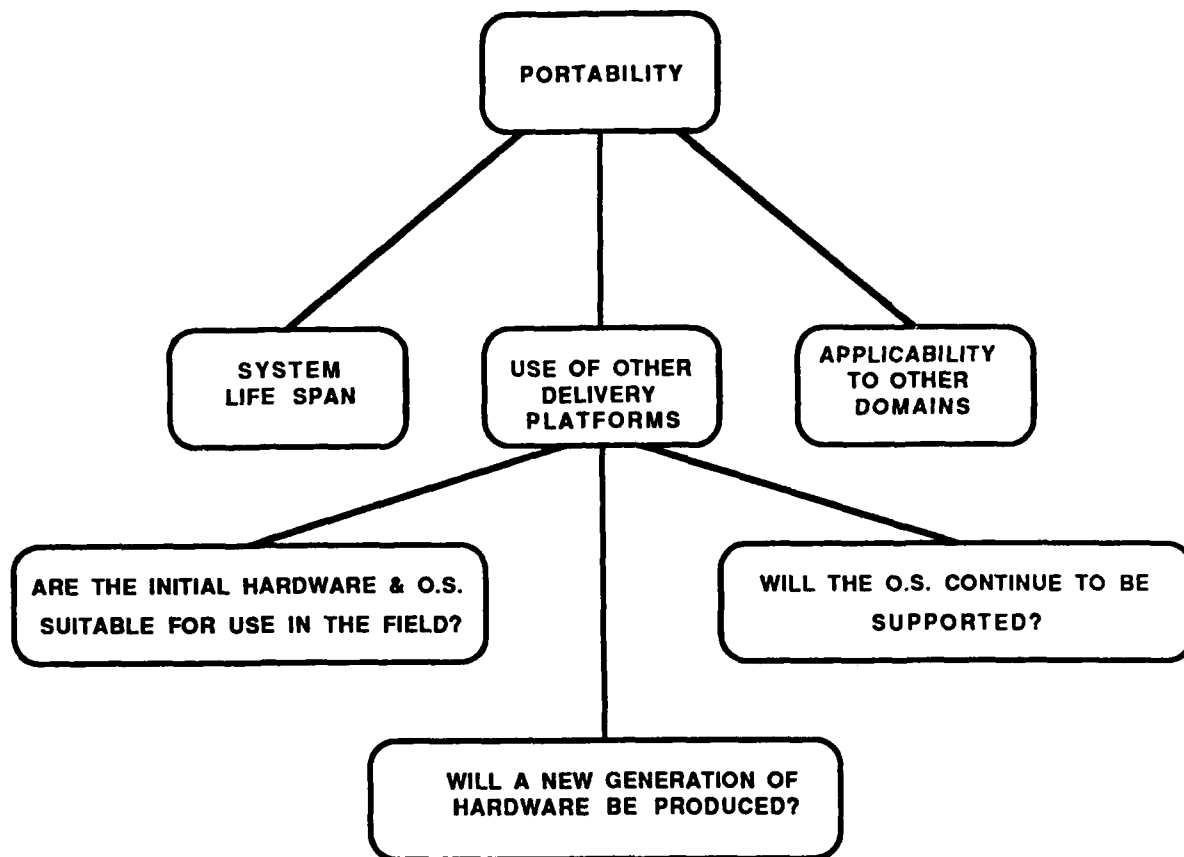


Figure 3.2.1.2-2 Sample Acquisition Attributes

3.2.1.2.1 Command and Control Quality Concerns

FINDING:

The process of accounting for Command and Control quality concerns is very general. The user is asked to refer to a table for suggestions of which quality factors are "important" for various command and control functions.

RECOMMENDATION:

More specific information is needed. In addition to the table, this paragraph should describe important quality considerations with respect to this type of system. For each factor, the text should discuss which command and control functions might have a requirement for high quality and why.

3.2.1.2.2 System Quality Factors

FINDING:

This paragraph adequately describes the relationship between system and software quality factors.

3.2.1.2.3 Quality Requirements Survey

FINDING:

The survey instructions are too brief to adequately guide someone who is unfamiliar with the RADC software quality framework. Definitions of the quality factors are at a high level, and may not be understood correctly. The meanings of "excellent", "good", and "average" are also open to interpretation. Certainly, different people will interpret them differently.

As described, each respondent can choose his own set of system functions. This will result in responses which cannot be compared with one another, and which cannot be used easily to draw conclusions about quality goals.

Finally, the text of this paragraph suggests that the respondents add an explanation of each entry in the table. This "justification" is very important in evaluating the responses, but the instructions do not specifically ask for such a response.

RECOMMENDATION:

The instructions should be greatly expanded in order to better define the software quality factors and the various goals. It would be especially useful if a rich set of examples were given to illustrate the kinds of results which would be expected for excellent, good, and average levels of quality for each factor.

This approach has the disadvantage that the survey would become much more demanding of the respondent. Much time would be consumed learning about the quality factors and attempting to "standardize" on the meanings of excellent, good, and average.

An alternative exists which would solve the problems of the current survey approach without this additional time-consuming learning curve for the respondent. This approach involves the use of the "acquisition framework" (see 3.2.1.2).

3.2.1.2.4 Complimentary Quality Factors

FINDING:

This paragraph suggests that several factor definitions are incomplete. If complimentary factors must be specified to produce "true indicators of the total quality present", then it seems that a single factor does not completely define the desired quality. For example, a high REUSABILITY score alone does not appear to mean that the software is indeed highly reusable. The complimentary factors of CORRECTNESS, MAINTAINABILITY and VERIFIABILITY must also receive high scores. This is very misleading, since each factor should be *completely* defined by its criteria.

RECOMMENDATION:

Redefine the relationships between factors and criteria to eliminate the need for complimentary factors. If software must be reliable, correct, maintainable, and verifiable in order to be considered reusable, then the criteria for REUSABILITY should include all these attributes.

3.2.1.2.5 Quality Goals Assignment

FINDING:

This paragraph discusses the initial assignment of quality goals based on command and control quality concerns, system quality factors, quality requirements survey, and complimentary quality factors. However, the example only shows where complimentary factor considerations modified the quality survey results. No instructions are given regarding how to include the other areas into the initial goal determination.

RECOMMENDATION:

This paragraph should describe the process for considering all relevant details when making this initial quality goals assignment. Earlier paragraphs result in specific quality factor considerations. These should be captured in a tabular format which represents all the information about a system function in terms of command and control quality concerns, system quality factors, all quality requirements survey responses, and complimentary factors. The table can then be used to develop one

goal for each factor. This approach brings together all information in one place. The table can be supported with notes about trade-offs and decisions related to the assignment of quality goals.

3.2.1.3 Consider Interrelationships (Step 3)

FINDING:

This paragraph is concerned with the technical feasibility of achieving the specified quality goals. The title "Consider Interrelationships" focuses on *how* the feasibility is examined, rather than *what* is being done.

The paragraph indicates that four areas are explored in accomplishing this step, corresponding with the four subparagraphs (4.1.3.1 - 4.1.3.4). The first two items, shared criteria and beneficial and adverse relationships, simply define concepts rather than describe steps to take. Actually, only quantification of relationships and review of quality goals represent actions.

RECOMMENDATION:

Rename this paragraph "Examine Technical Feasibility" to convey what is being explored.

This paragraph, and subparagraphs should be restructured to more clearly differentiate between definitions and procedural steps.

3.2.1.3.1 Shared Criteria

FINDING:

This paragraph defines shared criteria in terms of cost savings to achieve and to measure such criteria. The example and discussion of cost do not belong in this section.

This paragraph references Table 3.2-1, which requires the reader to leaf back through the previous section. This is not necessary since Table 4.1.3-1, on the next page, contains the same information.

RECOMMENDATION:

More detail could be provided to explain the technical ramifications of shared criteria. The example and cost discussion should be moved to paragraph 4.1.4, Consider Costs.

Change the table reference to 4.1.3-1 so the reader can easily find the list of shared criteria.

3.2.1.3.2 Beneficial and Adverse Relationships

FINDING:

This paragraph discusses the interrelationships among factors, both beneficial and adverse. The paragraph is intended to provide background information, not to define steps of the methodology to be followed.

RECOMMENDATIONS:

Add a statement to the effect that this paragraph presents background to enable the SAM to implement later steps in the quality specification process.

3.2.1.3.3 Quantification of Relationships

FINDING:

This paragraph treats a very complex subject with relatively little text. In general, more explanation would help clarify the concepts as well as the process being described.

Terminology is inconsistent regarding shared criteria. This paragraph, as well as the tables which it references, use the term "common" rather than "shared" criteria. This is confusing, particularly after an earlier paragraph defined "shared" criteria, and never referred to "common" criteria.

The example chooses to ignore the effects of shared criteria when quantifying the relationships among factors. This seems strange, particularly when the effects of shared criteria are shown in Table 4.1.3-2. The example says only that this was done to "remain conservative when estimating positive effects". No guidance is given regarding when to include or exclude these effects.

All positive factor interrelationships in Table 4.1.3-2 are shown to have an equal effect: low. In fact, the effects of shared criteria should be much stronger than the effects of other, secondary relationships. These positive, or cooperative relationships have to do with the ease in achieving the desired factor goals. Clearly, relationships which do not involve "essential characteristics" of two factors have a small effect on achieving the factor goals. Just the same, when factors share one or more criteria, the effect on effort to achieve quality goals is much stronger.

In quantifying the relationships among factors, no attempt is made to account for the importance (goal) assigned to the factors. Certainly, the effect of one factor on the others is greatest if the factor goal is excellent. This paragraph treats all factors equally, whether the goal is "good" or "excellent".

As previously mentioned, the text does not adequately discuss how to use Tables 4.1.3-2 and 4.1.3-3 to

quantify factor interrelationships. In addition, the tables themselves are difficult to use, primarily due to the intra-table referencing.

The text mentions Table 2.2-2, which can be found only by leafing through previous sections. This is not at all necessary, since the same information can be found in Table 4.1.3-4.

RECOMMENDATION:

Rewrite this paragraph to be more explanatory. Begin with a discussion of what is meant by quantification of relationships. Explain at a high level how the tables represent interactions between factors numerically. Describe the procedure for using the tables to compute values representing interrelationships. Finally, explain how to enter the results into Table 4.1.3-4.

Adopt a consistent terminology with respect to shared / common criteria. Use the same terminology in the text and the tables.

Clarify the role that shared criteria play in this procedure. If they are important, as they seem, then account for them in the example. If they should sometimes be ignored, explain when and why.

Alter Table 4.1.3-2 to emphasize shared criteria more strongly than other, secondary relationships among factors.

Modify the procedure for quantifying effects among factors to account for the effects of different factor goals. Factors with "excellent" goals will have stronger effects than factors with just "average" goals.

Eliminate references (see 4, above) from Tables 4.1.3-2 and 4.1.3-3 to improve the usability of these tables.

Eliminate the reference, in the text, to Table 2.2-2. This can be replaced with a reference to Table 4.1.3-4.

3.2.1.3.4 Review of Quality Goals

FINDING:

This paragraph explains how to modify the initial quality factor goals based on factor relationships. The paragraph briefly discusses some ramifications of conflicting relationships, but fails to explain how the cooperative, or beneficial, relationships should be taken into account. Based on this paragraph, it is not clear why it was necessary to quantify the cooperating relationships.

The example provided by this paragraph indicates that, based on strong negative relationships, "achieving the initial set of goals is not possible". This rather strong statement is not well supported in the text. It raises the question, how can one draw the line between impossible and merely difficult to achieve initial sets of quality goals? This paragraph does not provide sufficient detail to answer such questions.

RECOMMENDATION:

Additional information should be provided to expand on the role of beneficial relationships among factors. If initial quality goals are not adjusted based on the "positive totals" (see Table 4.1.3-4 in Volume II) then the text should be modified to eliminate the quantification of cooperative relationships.

It is likely that no clear line exists to distinguish technically difficult combinations of quality factor goals from those which would be "impossible" to achieve. If that is the case, then the example in this paragraph should be toned down. If no firm guidelines are possible, then this paragraph should emphasize the correlation between large negative interrelationship totals and technical difficulty of achieving those goals.

3.2.1.4 Consider Costs (Step 4)

Since the SBSDA project was complete before our quality specification process began, we did not formally evaluate the cost feasibility of our quality goals. The following paragraphs provide findings and recommendations based on a review of these procedures.

3.2.1.4.1 Life-Cycle Quality Costs and Benefits

FINDING:

This paragraph describes the life cycle cost ramifications of applying quality measurement technology to software development. The discussion is an important one, particularly for the SAM who must decide if the additional effort involved in the methodology is warranted. However, this paragraph attempts to quantify the cost effects in terms of percentage of overall cost for each development phase. The area of cost impacts, at this time, seems better suited to a subjective discussion, since no basis exists to define the relative cost impacts of each factor to each development phase. The text alludes to this subjectivity by saying that "baseline cost...reflects cost if the quality factor was not emphasized in the life cycle and only a nominal amount of quality was present." The tables then suggest a range of cost variation, expressed as a percentage, for each factor and for each life cycle phase. It seems that this "baseline cost" is highly variable. The effects of quality specification and evaluation could have a *profound or minimal effect on cost*, depending on the existing development standards and common practices of the

development team.

Another concern with Table 4.1.4-1 is that no basis is identified for the cost ranges. It appears that these ranges represent subjective feelings about the relative costs rather than accurate quantities, validated through experimentation.

Finally, the title of the paragraph, "Life-Cycle Quality Costs and Benefits", is less descriptive than it might be of the actual paragraph contents. The paragraph doesn't really address life cycle quality benefits. Instead, the paragraph explores the negative and positive cost impacts of the methodology.

RECOMMENDATIONS:

This paragraph presents good background on the cost implications of the methodology. The SAM must consider this before specifying quality factors and requiring the developer to evaluate quality as development proceeds. The discussion does not belong here, however. This paragraph more properly belongs in section 2, along with other material which describes the methodology.

The magnitudes of cost impacts are important, but must be validated. Until this has been done, these numbers should be expressed simply as relative magnitudes, not as a percentage of overall development costs. The text should specifically indicate the limitations of the values pending further validation.

In order to be more indicative of the paragraph contents, the title should be changed to "Life-Cycle Quality Cost Implications".

3.2.1.4.2 Cost Variation Estimates

FINDING:

This paragraph uses information contained in the previous paragraph to estimate the cost impact for each quality factor across the software development life cycle. The discussion is useful in the sense that it provides an overview of how costs may increase and decrease as a result of using the methodology. As mentioned above, until these numbers have been validated it is risky to use them in estimating actual impacts.

RECOMMENDATIONS:

Until these cost effects have been validated, they should not be applied by the SAM to estimate the cost effects on a specific project. This discussion should be moved along with the previous paragraph to section 2. It should also be toned down in order to emphasize that the numbers represent potential magnitudes at present, not actual percentages.

3.2.1.4.3 Cost Effects of Factor Interrelationships

FINDING:

This paragraph discusses the cost effects of negative and positive factor interrelationships. Once again, the discussion is useful in demonstrating to the SAM how the components of the framework can interact and how they may impact the cost of software development. However, as discussed above, no evidence has been given that these effects have been validated. It is of questionable value to accommodate the effects of factor interrelationships until a more sound basis has been demonstrated for the numbers.

This paragraph contains a brief mention of complimentary factors. It indicates that, unless the appropriate complimentary factors have been specified, the negative cost effects will be greater than shown here. While that is likely, it provides additional evidence that factor complementarity should be eliminated (see paragraph 3.2.1.2.4).

RECOMMENDATIONS:

This discussion also should be moved to section 2. In addition, the text should be made less definite. Until validation has been performed, this paragraph should simply describe these as relative effects.

3.2.1.4.4 Review of Quality Goals

FINDING:

This paragraph discusses two separate topics: modifying factor goals based on cost considerations, and expressing factor goals in numerical terms. It seems odd to combine these two topics into a single paragraph.

In discussing numerical values for factor goals, the paragraph does well to mention the lack of industry standards and the need to use judgement. However, little help is provided to the SAM in knowing what values or ranges might be appropriate in various circumstances. The paragraph lacks detail about the alternatives and the accompanying rationale.

RECOMMENDATIONS:

It has been previously recommended that the discussion of cost effects be moved to section 2. If that is done, the paragraph which describes adjusting factor goals based on cost considerations should be moved as well. This paragraph should once again emphasize the present, subjective nature of these cost considerations.

The remaining discussion regarding quantification of factor goals should be put into a separate paragraph. In keeping with the existing numbering convention, the new paragraph should be

4.1.5 Establish Numeric Factor Goals. This paragraph should be expanded to include a discussion of various alternatives for quantification. How do various applications affect the choice of goals? Perhaps the goals should remain fixed until more empirical evidence has been gathered to support various ranges for different applications.

3.2.2 Select and Specify Quality Criteria

FINDING:

This paragraph introduces the section on specifying quality criteria. It refers to three procedures, called Step 1, Step 2, and Step 3, but fails to label each step as was done in paragraph 4.1. Also, less information is given about these steps than was provided in paragraph 4.1. Finally, the term "select" in the paragraph title is not particularly applicable.

RECOMMENDATION:

Add the step number to the names of the three steps for clarity. Expand the discussion which introduces the purpose of each step. Change the paragraph title to "Identify and Specify Quality Criteria".

3.2.2.1 Select Criteria (Step 1)

FINDING:

The name of this step is somewhat misleading, since criteria are not being selected, *per sé*. Rather, as the text indicates, criteria are *identified* which are attributes of the factors for which goals were established.

RECOMMENDATION:

Rename this step "Identify Criteria (Step 1)".

3.2.2.2 Assign Weighting Formulas (Step 2)

FINDING:

This paragraph defines the purpose of and procedures for establishing criteria weighting formulas. Although the example hints at it, the text does not clearly explain that one set of these formulas is needed for each system function.

The formulas described by this paragraph, particularly in Table 4.2-1, are not directly useable for calculating factor scores from the worksheets. This is because the formulas allow for *all* criteria to receive a weighting. In reality, however, *no* worksheet measures *all* criteria. Further, tailoring could result in the elimination of criteria. The result is that these "ideal" formulas must be modified for use with each worksheet.

Aside from the example in the text, no help is given to identify "typical" examples of emphasizing or de-emphasizing criteria. This procedure can have a significant effect on factor scoring, and deserves more attention.

RECOMMENDATION:

The paragraph should more clearly explain the relationship between the weighting formulas, factors, worksheets, and system functions. A diagram would be helpful to illustrate the concepts.

After developing the "ideal" weighting formulas for each system function, a step is needed to transform these formulas for use with each worksheet. The procedure is not difficult, and involves adjusting the formula for each factor which has one or more criteria that are not measured by a particular worksheet. A series of blank forms (one per worksheet) could be provided which explicitly "zero out" the appropriate criteria. A coefficient could then be supplied with which to adjust the remaining entries, for each formula. Perhaps this translation could be made in conjunction with the quality evaluation procedure, and described in Volume III (see paragraph 3.3.3.1).

Decimal weighting coefficients are difficult to work with. For example, if a factor such as REUSABILITY is comprised of 9 criteria which are all weighted equally, the correct weighting is .111111... This number cannot be exactly represented by a decimal, but can be represented by the fraction $1/9$. The problem with decimal fractions is compounded if, for example, one of the criteria is to be slightly emphasized, while the remaining 8 remain equally weighted. This can be solved by the use of fractional coefficients, where the denominator represents the total "points" for all criteria (not necessarily 10). The numerator represents the portion of the total "points" allocated to any one criterion. This would avoid instances where particular (decimal) coefficients are chosen in weighting formulas because they are convenient, not necessarily because they represent the desired relationship among criteria. In other cases, decimal weighting coefficients are easier to specify. The text should provide examples of both, and the SAM can choose which approach best suits his needs. Finally, more detail should be given regarding when to emphasize or de-emphasize a criterion. Each criterion could be discussed, along with "typical" scenarios which might require other than an average weighting. The example only discusses two factors rather than presenting a general discussion of all factors and criteria.

3.2.2.3 Consider Interrelationships (Step 3)

FINDING:

This paragraph describes an example of adjusting the criteria weighting formulas based on positive and

negative relationships between criteria. Although a good example is shown, no real procedure for making this analysis is given. Also, the text does not explain what adjustments might be appropriate due to positive relationships.

RECOMMENDATION:

Much more explanation is needed in this paragraph to support this procedure. Specific steps should be described, along with forms and examples to support the text.

Once again, something must be said about the quantitative effects of positive relationships among criteria. If there are no quantitative effects, then the discussion of positive interrelationships should be removed from the guidebooks.

3.2.3 Select and Qualify Metrics

FINDING:

This paragraph describes procedures for identifying specific metrics and metric elements for use during the quality evaluation process. Not enough information is provided to explain why this process is necessary. Since the procedure consists of eliminating individual metrics and metric elements from consideration, use of the word "select" in the title of this paragraph is not appropriate.

RECOMMENDATION:

Add several statements, and perhaps an example, to explain why this is an important step in the quality specification process. Also, change the paragraph title to "Eliminate Metrics from Consideration".

3.2.3.1 Identify Metrics (Step 1)

FINDING:

This paragraph explains the process of identifying all metrics to be measured, based on criteria which have been given a non-zero weighting. Metrics will, in general, be measured for the entire system - not for one function at a time. Therefore, this selection process must account for the factors and criteria which will be measured for *all system functions*. As written, the text only mentions one example function, and does not explain the relationship of metrics to all system functions. This could be very misleading.

RECOMMENDATION:

Add a paragraph which explains the details of metric selection for the entire system rather than for a single function.

3.2.3.2 Select and Qualify Metric Elements (Step 2)

FINDING:

This paragraph describes the process for eliminating certain metric elements from consideration. As such, the title "Select..." is somewhat misleading.

This paragraph contains a lengthy example, but far too little explanation of the process involved in eliminating metric elements. The ramifications for the quality evaluation phase are not well explained.

With such a large number of metric elements (over 300), knowing which ought to be eliminated presents a special problem. The text provides no good guidelines for efficiently performing this step.

Finally, it is difficult, at best, to effectively eliminate metric elements for the entire development life cycle. At the concept definition phase, it is impossible to foresee the design decisions which will be made later. Without a good understanding of the design of the system, it is impossible to effectively eliminate unnecessary or irrelevant metric elements.

RECOMMENDATION:

The primary activity which is performed for this step is the *elimination*, rather than the selection, of metric elements. Accordingly, the paragraph title should be changed to "Eliminate Unnecessary Metric Elements".

The discussion of what is being done and why should be expanded. It should be explicitly explained that eliminating metric elements will avoid scoring biases later, when N/A scores can affect overall factor scores. It should be reiterated that this process is performed once for the entire system, not once per function.

To assist in identifying candidates for elimination, "typical" categories could be provided, along with the metric elements which "belong" to each category. The system in question is then considered with respect to each category. For each, if the system does not "fit" the category, all associated metric elements can be eliminated. Sample categories might be hard real-time systems, mainframe based systems, micro-based systems, database intensive systems, and systems which communicate with other, external systems. Further, if the "acquisition hierarchy" concept is adopted (see paragraph 3.2.1.2) the acquisition metrics could be related, through tables, with the existing metrics. These relationships could then be used to tailor the metrics based on the scores for important and unimportant acquisition concerns.

It is impractical to eliminate metric elements from the coding phase up front, during concept definition. Therefore, this step should be described as an iterative one. The guidebook should allow the acquisition manager to eliminate metric elements just prior to the application of the worksheets for a given phase.

3.2.4 Assess Compliance with Requirements

This paragraph introduces the activities involved in assessing software quality results against specifications at each life cycle phase. The following subparagraphs make up the process:

3.2.4.1 Review Requirements Allocations and Evaluation Formulas

FINDING:

This paragraph is a good description of the review process. It appears to provide a complete overview of the activity.

3.2.4.2 Review Factor Scores

FINDING:

This paragraph describes several activities involved in analyzing factor scores. First, it should be stressed that variations in scores across software entities (CSCIs, CSCs, units) should be explored, and this type of analysis is very useful. The developer and the acquisition manager can identify certain problems early with this approach. The other analyses described here, however, are not particularly helpful without a statistically valid method for normalizing the scores. Specifically, no basis exists to meaningfully compare factor goals with achieved scores. The same problem exists with comparing scores across worksheets. The text does not indicate what would be considered "acceptable variations" in scores and, indeed, no good definition can be proposed without a basis for normalizing factor scores.

The text states that "scoring should show an upward trend over development cycle". This seems incorrect for two reasons. First, as mentioned above, no basis exists *currently* for comparing scores across worksheets. We have no way to know that identical scores on successive worksheets represent equal quality levels. Second, even if we had normalized scores to work with, it is not always the case that scores ought to improve over the course of the development. If the scores on worksheet 0 are all at an acceptable level, then they *should not* improve on the subsequent worksheets. To do so would represent an inefficient use of project resources. "Better is the enemy of good enough..."

RECOMMENDATIONS:

The very useful aspect of evaluating scoring variations across software entities on a particular worksheet should receive more emphasis. All other comparisons of scoring should be eliminated until more statistical validation has been performed.

3.2.4.3 Review Criteria Scores

FINDING:

This paragraph appropriately stresses the benefits of reviewing variations of criteria scores across software entities. No changes are needed here.

3.2.4.4 Review Metric Scores

FINDING:

This paragraph, like the paragraph above, properly discusses the benefit of reviewing metric scores for variations. However, the paragraph suggests that, if a metric has a constant score (due, for instance, to a compiler characteristic) that the metric be dropped. Similarly, if a metric has a consistently high score due, perhaps, to "good development practices by the contractor", that metric also should be omitted from scoring calculations. It appears that, in the interest of avoiding perceived scoring biases, the text is suggesting that steps be taken to inject negative biases into the scoring. The rationale for this is not at all clear.

The paragraph contains a good wrap-up describing the beneficial visibility which scoring reviews provide.

RECOMMENDATIONS:

Rather than dropping such "constant" or "consistently high scoring" metrics (and lowering the scores as a result), it seems proper that the scores reflect such *beneficial effects on software quality*. These portions of the text should be deleted.

3.2.A Appendix A - Metric Worksheets

FINDING:

This Appendix contains no information.

RECOMMENDATION:

Delete this appendix from Volume II. Insure that all references to this appendix are changed to refer to Volume III.

3.2.B Appendix B - Factor Scoresheets

FINDING:

This Appendix contains no information.

RECOMMENDATION:

Delete this appendix from Volume II. Insure that all references to this appendix are changed to refer to Volume III.

3.2.C Appendix C - Software Quality Evaluation Report

FINDING:

This report is produced during the quality evaluation process, not as a result of quality specification activities. Therefore it should not appear in Volume II.

RECOMMENDATION:

Delete this appendix from Volume II. Insure that all references to this appendix are changed to refer to Volume III.

3.3 SOFTWARE QUALITY EVALUATION

This subsection identifies specific findings and recommendations which relate to guidebook volume III - Specifications of Software Quality Attributes - Software Quality Evaluation Guidebook. In particular, this subsection focuses on the methodology described by section 4 of the guidebook.

3.3.0 Software Quality Evaluation Methodology

FINDING:

This introduction to quality evaluation is generally complete and useful. The only difficulty might arise from its length (5 pages).

RECOMMENDATIONS:

Partition this paragraph into a more granular structure with subparagraph numbers and titles.

3.3.1 Identify Allocation Relationships

FINDING:

This paragraph discusses the procedures for weighting CSCI scores in the calculation of factor scores for each system function. No figures or tables are presented to assist with understanding this complex subject. Further, the process, as described, does not work in practice. This is because factor scores cannot be readily calculated for a CSCI which has quality requirements allocated from multiple system functions. Each system function might have a different criteria weighting formula. Which formula would be used, then, to calculate factor scores for the CSCI? The text does not address this complication. For further discussion of the overall scoring issues, see paragraph 3.3.1.

RECOMMENDATIONS:

This paragraph should provide more detail to assist with deriving allocation relationships and corresponding matrices. In particular, several examples should be given, along with accompanying tables. The simple case of one system function and one CSCI should be explored first. This should be followed with an example of one system function and multiple CSCIs. The next example should discuss multiple functions (2 or more) and a single CSCI. Finally, a complex example should be explained with multiple system functions and multiple CSCIs. Each function should have quality requirements allocated to each CSCI. Such a progression of examples will help the SAM to appreciate the complexity of this allocation process, and will prepare him for the task of scoring.

3.3.2 Apply Worksheets

This paragraph provides a good introduction into the steps involved in applying the worksheets to products of software development.

3.3.2.1 Prepare Worksheets (Step 1)

FINDING:

This paragraph explains the process of preparing the worksheets for each phase of software development. In general, this process is straightforward. The only special case is test and integration, and the text lacks adequate detail here. The rather cursory explanation of how to prepare worksheets for these phases is not adequate to guide the developer through the process.

The text explains that the system level specification will contain details about the use of factors, criteria and metrics for each system function. This is partly true, but the relevant data item description (DID:CMAN-80008) does not require that criteria and metrics be defined in this document. This leaves the developer, with guidance from the SAM, to tailor the worksheets. This tailoring will, of course, have a significant effect on the quality scores. Since these scores can be contractually binding, it is felt that the SAM, and not the developer, should have the final decision on this tailoring. Guidance is lacking regarding exactly how to accomplish this tailoring. How is it determined that questions are unneeded?

Finally, the text directs that a separate worksheet be used for each system function. This seems to be unnecessary, and a poor use of resources. Instead, a single worksheet can be used for the entire system, and one set of criteria scores developed. Separate factor scoring can then be performed for each system function, using the criteria weighting unique to each function.

RECOMMENDATIONS:

The text should be modified to separately address worksheets and tailoring procedures for each phase of the software life cycle. Special emphasis should be placed on the test and integration phases.

Additional assistance in tailoring the worksheets should be provided. Indicate that this tailoring must be approved by the SAM. Discuss the role which the quality surveys and the acquisition hierarchy can play in selecting important metrics. Change the discussion of the system specification document to more properly reflect its emphasis on factors rather than criteria and metrics.

Modify the approach to worksheet 0, such that the software development products only need to be evaluated one time, and only one worksheet 0 need be completed. Establish scoring procedures to produce separate factor scores for each system function.

3.3.2.2 Gather Source Material (Step 2)

FINDING:

This paragraph adequately describes the process of gathering the required source material, prior to completing a worksheet. More attention could be given, however, to the implications of working with specifications which do not follow DoD-STD-2167.

3.3.2.3 Answer Worksheet Questions (Step 3)

FINDING:

This paragraph describes the process of answering the worksheet questions. The underlying principle must be to produce unbiased, reproducible results. Many factors affect this; Following are several issues with the text.

Questions are organized strictly alphabetically. Although the text indicates that it should be "simple" for people familiar with the documentation scheme to find answers to the questions, that was not our experience. It is quite difficult to correlate the questions with the proper source material.

The examples and glossary contained in the worksheets are often inadequate to determine the precise intent of the question. This has obvious impacts on the repeatability of scores. The situation is compounded due to the varied backgrounds of reviewers.

Although the text encourages the reviewer to note relevant comments, including the source material used in answering questions, the worksheets provide no place to capture such information.

It was often difficult to determine whether "N/A" or "no" was the appropriate answer to questions. Clearly, the choice can have a profound impact on the resulting scores. It seems unwise to allow the reviewer, at his discretion, to mark an answer "N/A", and thus positively bias the scores.

Much time was wasted in reviewing questions which were related. The answers to such questions will always be "no" or "N/A" when the first question in the group is answered that way. In such cases, the worksheet should provide an aid to allow the reviewer to skip the remaining questions in the group. For an example of this problem, see CL.1(2) - CL.1(6) on worksheet 0.

RECOMMENDATIONS:

In order to more readily find answers to the questions, one of two solutions might be adopted. First, the questions on each worksheet might be organized by DID and by section number within each DID. That way, all questions which could be answered, for example, by reviewing the Software Development Plan would be collected together. If it is desired to maintain the current alphabetical sequence of questions, a cross-reference matrix could be produced containing the same information.

Each metric element should be accompanied with a background sheet which describes the metric element and provides the details necessary for its proper use. Such a reference would detail how it is to be used, what information is needed to measure it, how to compute the value, how to interpret the results, assumptions for use, training required to use the metric element, an example of its use and, importantly, references to literature for further information. This background sheet would also be useful during worksheet tailoring. The IEEE Software Quality Metrics Standard Committee, P1061, has produced a draft standard which addresses such a background sheet in more detail [18].

In order to facilitate the recording of relevant information, we found it beneficial to capture the answers to metric questions on a separate form (see Metric Element Scoring Form, Appendix D). This form has provisions for noting the source documentation used to answer the question, as well as relevant rationale. The use of such a form should be made mandatory to insure that questions about scores can be answered later.

The option of "N/A" should be eliminated from worksheets. This will encourage the SAM and developer to work more carefully during worksheet tailoring. It will also have the effect of minimizing biases due to the inappropriate use of "N/A" by the reviewers.

Related questions on the worksheets should be grouped or bracketed in some way. This will convey to the reviewers which questions should be skipped or answered alike, as appropriate.

3.3.3 Score Factors

FINDING:

This introduction adequately identifies the steps involved in scoring. The steps are explained in the following paragraphs.

3.3.3.1 Prepare Scoresheets (Step 1)

FINDING:

This paragraph addresses the process of tailoring scoresheets to reflect the specific factors, criteria, and weighting functions associated with scoring a worksheet. The text and the scoresheets fall short in a number of important ways.

First, it must be remembered that two types of weighting are involved in scoring. The criteria scores must be weighted when calculating factor scores. Also, scores for each CSCI must be weighted when computing scores for system functions. The scoresheets do not accomodate either form of weighting. Although the text mentions criteria weighting, the discussion *omits CSCI-to-function weighting*.

The criteria weighting formulas which were developed as part of quality specification do not provide for the fact that *not all criteria are measured on each worksheet*. Thus, in most cases, the coefficients applied to the remaining criteria *will not sum to 1.0*. This means that, using the "default" criteria weighting formulas, a factor cannot possibly receive a score of 1.

The scoresheets are simply not an adequate tool for scoring complex cases. As an example, consider how they might be used to represent quality scores of several system functions allocated (many to many) to several CSCIs. When computing the CSCI factor scores, from which system function would the criteria weighting formulas be used?

In addition to the above, several minor problems exist with the scoresheets. Scores for some criteria, e.g. SI, are calculated repeatedly for use with multiple factors. Also, the scoresheet for EFFICIENCY improperly sums all the metric scores to yield the factor score, without producing criteria scores.

Finally, the scoring scenario is not adequately defined for the test and integration phase. Appendix A indicates which metric elements should be re-measured during these phases. However, no guidance is given as to how to recompute criteria and factor scores. Paragraph 4.2.1 indicates that the answers "can be monitored individually and compared to answers from prior phases or can be used with applicable answers from prior phases to calculate new factor scores." The text does not explain which answers from which phases should be used.

RECOMMENDATIONS:

It is most important that the guidebook clearly and comprehensively treat the topic of scoring. Both simple and complex cases must be accommodated. A series of examples which illustrate various combinations of functions and CSCIs are necessary. To accomplish this, the existing scoresheets and accompanying text must be totally reworked.

First, a series of **criteria scoresheets** should be developed (see Figure 3.3.3.1-1). These are similar to the existing scoresheets, but only compute criteria values. This addresses the current problem of computing certain criteria values multiple times. The **criteria scoresheets** require minimal tailoring and can be used with each worksheet. For worksheet 0, only one **criteria scoresheet** is needed. For worksheets 1, 2, 3A, and 4A one **criteria scoresheet** per CSCI is required.

Criteria scores from the **criteria scoresheets** are used to calculate factor scores. On worksheet 0, this is fairly straightforward. The criteria scores are entered into a **function factor scoresheet** along with the criteria weighting formula for each factor (see Figure 3.3.3.1-2). The resulting computation yields the factor scores for the system function. A separate **function factor scoresheet** is completed for each system function.

On worksheets 1, 2, 3A, and 4A, criteria scores from multiple CSCIs must be combined to form criteria scores for each system function. This will be accomplished with a **function criteria scoresheet** (see Figure 3.3.3.1-3). This scoresheet allows the contribution of each CSCI to the function to be accounted for. A **function criteria scoresheet** will be completed for each system function. The criteria scores from each CSCI's **criteria scoresheet** are entered into the **function criteria scoresheet**. The CSCI weighting coefficient is entered for each CSCI. Finally, criteria scores are computed for each criteria which has been specified for this system function.

The resulting criteria scores on the **function criteria scoresheet** must be used to calculate factor scores. The function factor scoresheet may again be used for this purpose. Again, a separate **function factor scoresheet** is completed for each system function.

This process sounds rather complex, and indeed it is. Unfortunately, without this complexity, the scoring scenario fails to account for the possible variations in scoring. An overview of this process is presented in Figure 3.3.3.1-4.

Two possible solutions exist to accommodate scoring when a criterion is not measured on a worksheet (or is tailored out). One approach is to compute the factor score by simply dropping the missing criterion. The factor score can then be normalized to 1.0 by dividing the result by (1-criteria weight) for the

criterion which was dropped. A better approach, which was described in paragraph 3.2.2.2, is to account for the absence of criteria during quality specification, when the criteria weighting formulas are first derived. This approach involves producing a criteria weighting matrix for each factor. This matrix establishes a separate criterion weighting formula for each worksheet. Thus, criteria which aren't measured on a particular worksheet can be omitted from the matrix, and any criteria which are tailored out can be given coefficients of 0. Coefficients of the remaining criteria should sum to 1.

CRITERIA SCORESHEET

DATE: _____

WORKSHEET NO.: _____

CSCI: _____

SCORER: _____

METRIC ELEMENTS		METRIC SCORE		CRITERION SCORE CONSISTENCY ____
CS.1(1)	_____	CS.1	_____	
CS.1(2)	_____			
CS.1(3)	_____			
CS.1(4)	_____			
CS.1(5)	_____			
METRIC ELEMENTS		METRIC SCORE		
CS.2(1)	_____	CS.2	_____	
CS.2(2)	_____			
CS.2(3)	_____			
CS.2(4)	_____			
CS.2(5)	_____			
CS.2(6)	_____			

Figure 3.3.3.1-1 Criteria Scoresheet

FUNCTION FACTOR SCORESHEET

DATE: _____ SCORER: _____
 FUNCTION: _____

CORRECTNESS

	WS 0 WT * SCORE	
CRITERIA		
ACCURACY	* _____	= _____
COMPLETENESS	* _____	= _____
CONSISTENCY	* _____	= _____
TRACEABILITY	* _____	= _____
VISIBILITY	* _____	= _____
FACTOR SCORE		_____

	WS 1 WT * SCORE	
CRITERIA		
ACCURACY	* _____	= _____
COMPLETENESS	* _____	= _____
CONSISTENCY	* _____	= _____
TRACEABILITY	* _____	= _____
VISIBILITY	* _____	= _____
FACTOR SCORE		_____

	WS 2 WT * SCORE	
CRITERIA		
ACCURACY	* _____	= _____
COMPLETENESS	* _____	= _____
CONSISTENCY	* _____	= _____
TRACEABILITY	* _____	= _____
VISIBILITY	* _____	= _____
FACTOR SCORE		_____

	WS 3A WT * SCORE	
CRITERIA		
ACCURACY	* _____	= _____
COMPLETENESS	* _____	= _____
CONSISTENCY	* _____	= _____
TRACEABILITY	* _____	= _____
VISIBILITY	* _____	= _____
FACTOR SCORE		_____

	WS 4A WT * SCORE	
CRITERIA		
ACCURACY	* _____	= _____
COMPLETENESS	* _____	= _____
CONSISTENCY	* _____	= _____
TRACEABILITY	* _____	= _____
VISIBILITY	* _____	= _____
FACTOR SCORE		_____

Figure 3.3.3.1-2 Function Factor Scoresheet

FUNCTION CRITERIA SCORESHEET

DATE: _____ SCORER: _____
FUNCTION: _____

CRITERION	CSCI 1 WT*SCORE=	CSCI 2 WT*SCORE=	CSCI 3 WT*SCORE=
ACCURACY	_____	_____	_____
ANOMALY MANAGEMENT	_____	_____	_____
AUTONOMY	_____	_____	_____
COMPLETENESS	_____	_____	_____
CONSISTENCY	_____	_____	_____
DOCUMENT ACCESSIBILITY	_____	_____	_____
FUNCTIONAL SCOPE	_____	_____	_____
GENERALITY	_____	_____	_____
INDEPENDENCE	_____	_____	_____
MODULARITY	_____	_____	_____
OPERABILITY	_____	_____	_____
RECONFIGURABILITY	_____	_____	_____
SELF-DESCRIPTIVENESS	_____	_____	_____
SIMPLICITY	_____	_____	_____
SYSTEM CLARITY	_____	_____	_____
TRACEABILITY	_____	_____	_____
VISIBILITY	_____	_____	_____

CRITERION	CSCI 4 WT*SCORE=	CSCI 5 WT*SCORE=	CSCI 6 WT*SCORE=	SUM OF CSCI SCORES
ACCURACY	_____	_____	_____	_____
ANOMALY MANAGEMENT	_____	_____	_____	_____
AUTONOMY	_____	_____	_____	_____
COMPLETENESS	_____	_____	_____	_____
CONSISTENCY	_____	_____	_____	_____
DOCUMENT ACCESSIBILITY	_____	_____	_____	_____
FUNCTIONAL SCOPE	_____	_____	_____	_____
GENERALITY	_____	_____	_____	_____
INDEPENDENCE	_____	_____	_____	_____
MODULARITY	_____	_____	_____	_____
OPERABILITY	_____	_____	_____	_____
RECONFIGURABILITY	_____	_____	_____	_____
SELF-DESCRIPTIVENESS	_____	_____	_____	_____
SIMPLICITY	_____	_____	_____	_____
SYSTEM CLARITY	_____	_____	_____	_____
TRACEABILITY	_____	_____	_____	_____
VISIBILITY	_____	_____	_____	_____

Figure 3.3.3.1-3 Function Criteria Scoresheet

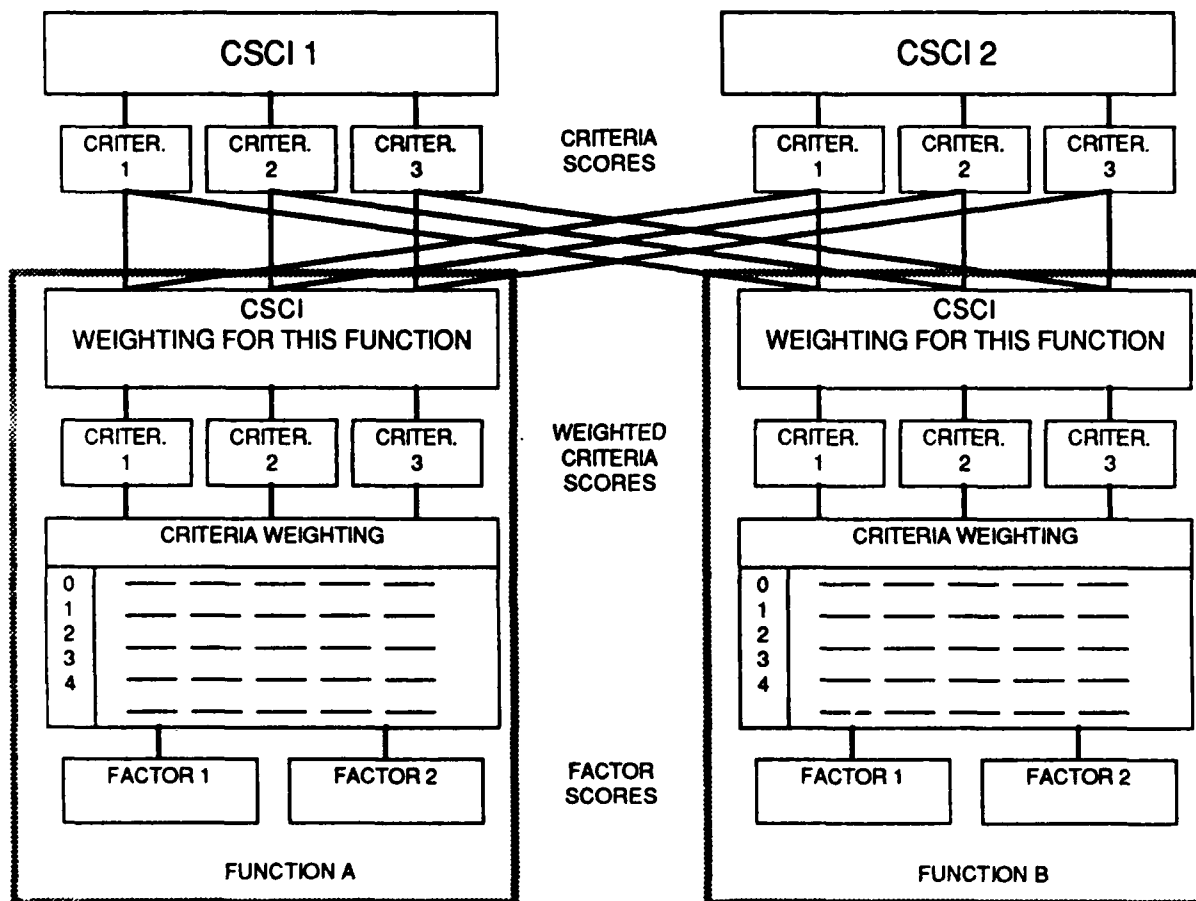


Figure 3.3.3.1-4 Scoring Process Overview

3.3.3.2 Calculate Factor Scores (Step 2)

FINDING:

Paragraph 3.3.3.1 identified several fundamental problems with the scoring process. These same problems affect this paragraph, but are not repeated here. This paragraph simply explains the process of entering numbers into the scoresheets and computing the results. Once the above problems have been solved and the scoresheets properly reflect correct scoring procedures, the specifics of how to fill in the blanks must be addressed.

3.3.4 Analyze Scoring

FINDING:

This paragraph introduces the steps involved in analyzing the scoring results. The text is adequate as it exists.

3.3.4.1 Calculate Functional Scores (Step 1)

FINDING:

As described in paragraph 3.3.3.1, the current scoring scenario has serious flaws. This paragraph describes the process for calculating factor scores for each system function. The problems with this have been discussed above, and need not be repeated here.

3.3.4.2 Compute Scoring Trends (Step 2)

FINDING:

This paragraph explains the procedures for analyzing scores across development phases in order to identify trends. A fundamental problem with this analysis is that no basis exists currently to compare scores from one worksheet to the next. The validity of trends over life cycle phases when different metric elements are scored is highly questionable.

RECOMMENDATIONS:

Until statistical validation or normalization has been done, this analysis should be eliminated from the methodology. Developers and SAMs should not expect to draw meaningful conclusions about the changes in software quality over the life of the program. Instead, the scores should be analyzed to highlight potential problem areas early in the development process. This can be done by looking for software entities which score markedly lower than the average. This analysis should be performed against the results from each worksheet rather than from one worksheet to the next. In addition to factor scores, metric and criteria scores can be useful.

3.3.4.3 Compare Scores with Requirements (Step 3)

FINDING:

This paragraph describes analyses of specified factor goals against measured scores. For the same reasons as stated above, the correlation of factor scores with quality factor ratings is highly questionable.

RECOMMENDATIONS:

Again, the analysis procedures should not suggest comparing quantities without a basis for normalization. We do not, for example, know how a measured SURVIVABILITY score of .7 compares with a specified goal of .8 - .9. We cannot conclusively say that we did or did not succeed in achieving the desired level of SURVIVABILITY. This paragraph should be eliminated.

3.3.4.4 Analyze Variations (Step 4)

FINDING:

This paragraph is concerned with identifying the causes of scoring deficiencies. As discussed above, it is unwarranted to compare specified goals against achieved results. However, scoring deficiencies can be noted by comparing the scores of multiple software entities on a single worksheet. As an example, one could discover that, of four CSCIs, one scored significantly lower on MODULARITY. The principles discussed in this paragraph would then be useful in identifying the cause of such a variation.

3.3.5 Recommend Corrective Actions

FINDING:

This paragraph introduces the steps for recommending corrective action. It is not clear why the two steps identified below were separated from the preceding discussion.

RECOMMENDATIONS:

Rather than introducing a separate subsection (4.5), the following subparagraphs and the steps they describe should be made part of the preceding discussion.

3.3.5.1 Summarize Problems (Step 1)

FINDING:

The process of categorizing problems is well stated.

RECOMMENDATIONS:

Renumber this paragraph as 4.4.5.

3.3.5.2 Provide Recommendations (Step 2)

FINDING:

The process of suggesting solutions to the problems is well stated.

RECOMMENDATIONS:

Renumber this paragraph as 4.4.6.

3.3.6 Automation

FINDING:

This discussion of potentials for automation adds nothing to the methodology.

RECOMMENDATIONS:

Delete this paragraph.

3.3.A Appendix A - Metric Worksheets

Findings for this appendix are documented elsewhere. See paragraph 3.3.2.

3.3.B Appendix B - Factor Scoresheets

Findings for this appendix are documented elsewhere. See paragraph 3.3.3.

3.3.C Appendix C - Software Quality Evaluation Report

FINDING:

This DID defines a very useful report. It is important for the developer to have a formal mechanism to communicate quality information to the SAM. However, two areas of the DID should be modified according to concerns previously discussed.

First, section 3.2 of the DID requires that quality requirement allocation relationships be described. It is appropriate to describe the contribution of each CSCI to each system function. This is in keeping with the allocation described in Volume III, paragraph 4.1. However, this allocation need not be discussed below the CSCI level, since mapping from units to CSCIs is done "automatically" when using worksheets 3B to complete 3A and worksheets 4B to complete 4A. Further, the DID treats this allocation as if *quality factor scores at the CSCI level* must be used to compute factor scores at the function level. As discussed above, this simple approach to scoring has serious shortfalls. The proposed scoring approach uses CSCI *criteria* scores to compute function *criteria* scores, according to a CSCI-to-function weighting matrix. This seems quite appropriate, since criteria, by definition, represent software attributes. It is, therefore, natural to compute scores for these software attributes at the CSCI level, and leave the

acquisition oriented factors for the system function level.

The second problem with the DID concerns section 3.4 - Data Analysis. As mentioned previously, such analyses should not be proposed until statistical validation and normalization functions have been accepted.

RECOMMENDATIONS:

Paragraph 3.2 of the DID should define the allocation relationships in terms of CSCI contribution to system functions without specifying that factor goals are being allocated to the CSCIs. Adopt an allocation philosophy which is consistent with the proposed scoring scenario.

Paragraph 3.4 in the DID should focus on those scoring variations for which the methodology is well suited. See paragraph 3.3.4 for further details.

SECTION 4 CONCLUSIONS

This section presents our major conclusions. Principally, the framework and methodology form a useful approach to software quality. Still, the methodology, in the form of the guidebooks, needs to be refined, and the framework must continually evolve.

In the software acquisition process, far too little emphasis has been placed on software quality. This has been due, in part, to the traditional practice of testing for quality rather than designing it in. Quality specification is beginning to receive more attention, particularly with the acceptance of DoD-STD-2167. The RADC software quality framework provides much needed tools for satisfying the requirements of 2167 to specify software quality.

Using this methodology also brings an early focus to overall software life cycle concerns. In the past, without such a structured approach to quality specification little attention was given to issues of software portability, adaptability, or maintainability. Now, appropriate levels of these attributes can be specified at the beginning of the system life cycle.

The most important benefits of the framework and methodology result from the ability to discover quality-related problems early during the development process, when such problems are least expensive to correct.

The guidebooks provide a useable tool for learning about the methodology. However, their utility could be greatly improved by separating theory and examples from procedures, and expanding the procedural discussions. In particular, a "cookbook" style would be appropriate for defining procedures to be followed. These steps would significantly increase the accessibility of the methodology to those acquisition managers and contractors who are not familiar with it.

The framework is a good paradigm for viewing software quality from different perspectives. To remain useful, the framework must evolve as the technology for developing software advances. Specifically, the revision of metric elements, metrics, and criteria definitions should be a continuous process. Acceptance of the framework will be based, in part, on the degree that it accounts for new techniques, practices, and procedures.

APPENDIX A

REFERENCES

- [1] *Software Quality Measurements Demonstrations, Statement of Work*, Rome Air Development Center, Griffiss Air Force Base, NY, PR No. B-5-3253, Contract No. F30602-85-C-0180, October 15, 1984
- [2] McCall, J.A., Richards, P.K., and Walters, G.F., *Factors in Software Quality*, Report No. RADC-TR-77-369, Rome Air Development Center, Griffiss Air Force Base, NY, November, 1977
- [3] McCall, J.A., and Matsumoto, M.T., *Software Quality Metrics Enhancements*, Report No. RADC-TR-80-109, Rome Air Development Center, Griffiss Air Force Base, NY, April, 1980
- [4] Bowen, T.P., et.al., *SW Interoperability & Reusability*, Report No. RADC-TR-83-174, Rome Air Development Center, Griffiss Air Force Base, NY, July, 1983
- [5] Bowen, T.P., et.al., *Software Quality Measurement for Distributed Systems*, Report No. RADC-TR-83-175, Rome Air Development Center, Griffiss Air Force Base, NY, July, 1983
- [6] Bowen, T.P., Wagle, G.B., and Tsai, J.T., *Specification of Software Quality Attributes*, Report No. RADC-TR-85-37, Rome Air Development Center, Griffiss Air Force Base, NY, February, 1985
- [7] *Senior Battle Staff Decision Aids*, Statement of Work, Rome Air Development Center, Griffiss Air Force Base, NY, PR No. B-3-3603, Contract No. F30602-83-C-0154, December 2, 1982
- [8] McIntyre, J.R., and Adelman, L., *Senior Battle Staff Decision Aids: Final Technical Report*, Report No. PAR 85-111, Rome Air Development Center, Griffiss Air Force Base, NY, December, 1985
- [9] Midwood, C., *RAA Software Quality Requirements Report*, Informal Report to RADC for Contract No. F30602-85-C-0180, April 24, 1986 (not published)
- [10] Warthman, J., *EPAA Software Quality Requirements Report*, Informal Report to RADC for Contract No. F30602-85-C-0180, April 4, 1986 (not published)
- [11] Warthman, J., and Midwood, C., *Software Quality Measurement Demonstration Implementation and Validation Plan*, Technical Report to RADC for Contract No. F30602-85-C-0180 (CDRL A002), December 20, 1985 (not published)
- [12] *Defense System Software Development*, DoD-STD-2167, June 4, 1985
- [13] *DoD Trusted Computer System Evaluation Criteria*, DoD-STD-5200.28, December, 1985
- [14] Goldberg, A., and Robson, D., *Smalltalk-80 The Language and its Implementation*, Addison-Wesley Publishing Co., Reading, MA, 1983
- [15] BYTE Magazine, Vol. 6, No. 8, June, 1981

- [16] Macintosh Developers Group, *Inside Macintosh: Macintosh User Interface Guidelines*, Apple Computer, Inc., Cupertino, CA, 1985
- [17] Parnas, D.L., *When can Software be Trustworthy?*, Paper presented to Fourth Annual Pacific Northwest Software Quality Conference, Portland, OR, November, 1986
- [18] IEEE Quality Metrics Standard Committee, P1061, *A Draft Standard for Software Quality Metrics*, Version 12, January 5, 1987

APPENDIX B

GLOSSARY

This glossary presents a list of terms, acronyms and abbreviations used in the report, along with their meanings.

AMS	Automated Measurement System
AMT	Automated Measurement Tool
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CPM	Critical Path Method - used for scheduling tasks and subtasks, and showing dependencies among them
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSI	Computer Science Innovations
DBMS	Data Base Management System
DFD	Data Flow Diagram
DID	Data Item Description
DoD	U.S. Department of Defense
DoD-STD-SDS	Software Development Standard (a.k.a. DoD-STD-2167)
ECOAEA	Enemy Course of Action Evaluation Aid
EPAA	Enemy Performance Assessment Aid
ESCMA	Enemy Sortie Capability Measurement Aid
FCA	Functional Configuration Audit
GUIDEBOOK	Software Quality Specification Guidebook, Volumes II and III
HWCI	Hardware Configuration Item
IV&V	Independent Verification and Validation
JTF	Joint Task Force

MIS	Management Information System
OCD	Operational Concepts Document
PARC	Palo Alto Research Center (XEROX)
PCA	Physical Configuration Audit
PDR	Preliminary Design Review
QA	Quality Assurance
QM	Quality Metrics
QMT	Quality Measurement Technology
QPT	Quality and Productivity Tool
RAA	Resource Apportionment Aid
RADC	Rome Air Development Center
SAM	Software Acquisition Manager
SBSDA	Senior Battle Staff Decision Aids
SDR	System Design Review
SOW	Statement of Work
SQM	Software Quality Measurement
SQMD	Software Quality Measurement Demonstration
SQRR	Software Quality Requirements Report
SRR	System Requirements Review
SRS	Software Requirements Specification
SSR	Software Specification Review
STP	Software Test Plan
STR	Software Test Report
TAF	Tactical Air Force
TCB	Trusted Computing Base
TIM	Technical Interchange Meeting
TRR	Test Readiness Review

APPENDIX C

QUALITY REQUIREMENTS SURVEY RESPONSES

This glossary presents the responses to the quality requirements surveys. Three individuals were asked to complete the survey: the SBSDA acquisition manager and two contractor personnel who were involved in the SBSDA development effort. For purposes of anonymity, the respondents were labelled 'A', 'B', and 'C'.

These individuals were asked to indicate the importance of each of thirteen software quality factors, with respect to the primary software functions. Answers were captured in a matrix. Also, each was asked to provide his rationale for the ratings. This was to be a short statement to explain why a particular level of importance was chosen. These responses were used, along with other information, to establish initial quality factor goals for each software function. Note that the each respondent addressed both EPAA and RAA.

SOFTWARE QUALITY FACTOR	PERFORMANCE					DESIGN			ADAPTATION				
	E F F I C I E N C Y	I N T E G R I T Y	R E L I A B I L I T Y	S U R V I V A B I L I T Y	U S A B I L I T Y	C O R R E C T N E S S	M A I N T A I N A B I L I T Y	V E R I F I A B I L I T Y	E X P A N D A B I L I T Y	F L E X A B I L I T Y	I N T E R O P E R A B I L I T Y	P O R T A B I L I T Y	R E U S A B I L I T Y
SYSTEM OR SOFTWARE- UNIQUE FUNCTION													
DEVELOP KNOWLEDGE	E	I	A	A	E	A	E	E	E	I	I	E	A
MONITOR ENEMY PERFORMANCE	E	I	A	A	E	A	E	E	E	I	I	E	A
ANALYZE ENEMY PERFORMANCE	E	I	A	A	E	A	E	E	E	I	I	E	A

NOTE FOR GOAL ENTRIES:

E = EXCELLENT

G = GOOD

A = AVERAGE

BLANK OR N/A =
NOT IMPORTANT OR
NOT APPLICABLE

QUALITY GOALS FOR THE EPAA DECISION AID

AD-A193 429

SOFTWARE QUALITY MEASUREMENT DEMONSTRATION PROJECT (I)
(U) COMPUTER SCIENCES INNOVATIONS PALM BAY FL
J L WARTHMAN DEC 87 RADC-TR-87-247 F30602-85-C-0180

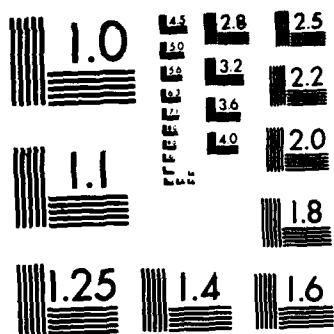
2/2

UNCLASSIFIED

F/G 12/5

NL





G MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

SOFTWARE QUALITY FACTOR	PERFORMANCE					DESIGN			ADAPTATION			
	E F F I C I E N C Y	I N T E G R I T Y	R E L I A B I L I T Y	S U R V I V A B I L I T Y	U S A B I L I T Y	C O R R E C T N E S S	M A I N T A I N A B I L I T Y	V E R I F I A B I L I T Y	E X P A N D A B I L I T Y	F L E X A B I L I T Y	I N T E R O P E R A B I L I T Y	P O R T A B I L I T Y
SYSTEM OR SOFTWARE- UNIQUE FUNCTION												
DEVELOP & ANALYZE AIR APPORTIONMENT RECOMMENDATION	E	A	A	A	E	A	E	E	G	A	E	A
PROVIDE RATIONALE & SUPPORT FOR RECOMMENDATION	E	A	A	A	E	A	E	E	G	A	E	A

NOTE FOR GOAL ENTRIES:

E = EXCELLENT
G = GOOD
A = AVERAGE
BLANK OR N/A =
NOT IMPORTANT OR
NOT APPLICABLE

QUALITY GOALS FOR THE RAA DECISION AID

Rationale for Quality Goal ratings:

RAA:

Efficiency - We wanted the Decision Aid to be implemented on an inexpensive; i.e. small machine so that we could get widespread utilization of a successful result at an affordable price. Consequently, based on prior experience, we thought it advisable to tightly constrain the designer or developer (for example, we have seen unconstrained decision aid developments take the full resources of a Vax 11/780, when in retrospect, a functionally equivalent solution, might otherwise, if constrained, have been implementable on a micro). This reasoning was prevalent because of two other considerations: (1) The RAA had the potential for subsequent integration with several other aids under development; however, the practical extent of the integration would be limited by the resources that could be put in small space, e.g. a microvax size machine; and (2) we had no idea what additional resource requirements would be needed in scaling up from laboratory prototypes to applications responsive to TAF field use requirements. Hence, the push be "anti Parkinson's Law" at the onset, i.e. very parsimonious.

Usability - Users would run the aid in support of a small percentage of their days work; would turn over very frequently; might not receive highly specialized training; would not be required to be computer literate, etc. Built in job training was called out as a design goal. Also, the aid must fit the problem solving/cognitive style of people who do the jobs/tasks that the decision aids are designed to assist.

Verifiability - A method was designed and previously verified as effective to measure the impact of the aid on the person, on the job, on the environment. Commitment to use that method as a means of verifying the utility of the aid was a given/constraint. That method required formal and informal evaluation by potential users in two cycles: first, by technical people simulating users; and secondly, by operational Air Force people who do the target task in the field today. Between the two cycles was a brief period within which the developers could fix as many as possible of the apparent deficiencies or limitations perceived by the first group. Consequently, there was an implied software verifiability requirement.

Flexibility - There was an explicit design goal to be able to readily adapt the product for use within different theaters of operation (and by implication at more than one command level).

Interoperability - The aid was to be designed so that in subsequent stages of development or operation, the aid would be tied to operational data base management systems for the input data. Also, in case of RAA, it was conceived of as a pre-planning (top level/preliminary) step to the Air Tasking Order (ATO) generation support task (detailed planning) against which another decision aid development effort was directed. Consequently, it was foreseen that the two aids (RAA and TEMPLAR) would eventually be linked or integrated.

Portability - At the time of program formulation, it was virtually certain that the machine on which the aid was developed, whatever the choice, would not be the machine(s) on which it would be implemented in the field since TAC had not yet decided on the TAF small computer(s). It was also possible that the aid might be fielded as another application on a larger machine. Hence, the requirements for high level language, modularity, etc. with possible portings to Perken-Elmer, VAXs, Cromemcos, and possibly Symbolics, for example.

Rationale for Quality Goal ratings:

EPAA:

Efficiency - In all honesty, this goal was pure "gut feel" - one which we could not adequately rationalize. Only very indirectly can I expand on this. The process which the aid was directed at was not well defined, subjective, and heavily dependent on large quantities of diverse data - much of which might not readily be accessible. Also, consideration might be given to the Expert System approach applied to define such a subjective process. To the extent that one can consider the front end knowledge engineering work to be part of software design and part of eventual adaption of the product to different commanders and their support staffs, "efficiency" was an important consideration.

Usability - See comment on this goal for RAA.

Verifiability - See comment on this goal for RAA.

Maintainability - Gut feel. Similar comment as pertains to efficiency. We assumed that this would be one of the riskiest of the four aids - might require more evolutionary development, hence more likely to need changes/fixes, etc.

Interoperability - Although it, like the other Senior Battle Staff aids, was to be initially built as a stand alone decision aid, the contractors were asked at the beginning of the program to consider eventual linking of the aids as a highly likely follow-on effort. Also, prior to completion of the implementation, it was made known that subsequent porting of the product from a VAX to a PC might happen to facilitate certain experimentation on group decision making.

Portability - see comment above. See also Portability comment in re RAA. Also, EPAA was seen as a potential component of an Integrated Aid set other than (just) the Senior Battle Staff Aids, e.g. as part of a suite of Intelligence Analysis support aids that feed the update to an Intelligence Data Handling System (which is one of the primary sources of aids that feed other aids-like the RAA).

Other comments on other factors for both EPAA and RAA.

1. The SBS program was 6.2-Exploratory Development. On successful accomplishment of proof of concept, the other factors would take on higher weights based on follow-on Advanced Development wherein performance considerations would be looked at intensively.

2. The follow-on to the SBS follow-on call for product/end-item evaluation by using appropriate metrics from the RADC S/W Quality Metrics Program. Hence, the SBS follow-on effort requires examination of metrics technology and the goals of program that will follow it in order to specify which metrics will be specified.

SOFTWARE QUALITY FACTOR SYSTEM OR SOFTWARE- UNIQUE FUNCTION	PERFORMANCE					DESIGN		ADAPTATION				
	E F F I C I E N C Y	I N T E G R I T Y	R E L I A B I L I T Y	S U R V I V A B I L I T Y	U S A B I L I T Y	C O R R E C T N E S S	M A I N T A I N A B I L I T Y	V E R I F I A B I L I T Y	E X P A N D A B I L I T Y	I N T E R O P E R A B I L I T Y	P O R T A B I L I T Y	R E U S A B I L I T Y
MONITORING Sys	E	E	E	G	E	G	G	E	G	G	G	G
ANALYZING Sys	G	E	G	E	E	G	G	G	G	A	A	G
DEVELOP KNOWLEDGE	G	G	A	E	E	E	E	E	E	B	G	E
SYSTEM												

NOTE FOR GOAL ENTRIES:

E = EXCELLENT

G = GOOD

A = AVERAGE

BLANK OR N/A =

NOT IMPORTANT OR

NOT APPLICABLE

Functions and Quality Factors for the

ENERGY PERFORMANCE ASSESSMENT NID

SOFTWARE QUALITY FACTOR	PERFORMANCE					DESIGN		ADAPTATION				
	E F F I C I E N C Y	I N T E R F A C E	R E L I A B I L I T Y	S U R V I V A B I L I T Y	U S A B I L I T Y	C O R R E C T N E S S	M A I N T E N A B I L I T Y	V E R I F I C A B I L I T Y	E X P A N D A B I L I T Y	I N T E R O P E R A B I L I T Y	P O R T A B I L I T Y	R E U S A B I L I T Y
SYSTEM OR SOFTWARE- UNIQUE FUNCTION												
Assist in												
Developing Aid												
Assess Reconn.	E	E	A	G	E	G	G	G	G	G	A	G
Provide Recon.												
• Briefing Reports												
For Appor. Conc	G	E	A	E	G	G	G	E	G	G	A	A

NOTE FOR GOAL ENTRIES:

E = EXCELLENT
G = GOOD
A = AVERAGE
BLANK OR N/A =
NOT IMPORTANT OR
NOT APPLICABLE

Functions and Quality Factors for the
Resource Apportionment Aid.

SOFTWARE QUALITY FACTOR SYSTEM OR SOFTWARE- UNIQUE FUNCTION	PERFORMANCE					DESIGN		ADAPTATION				
	E F F I C I E N C Y	I N T E G R I T Y	R E L I A B I L I T Y	S U R V I V A B I L I T Y	U S A B I L I T Y	C O R R E C T N E S S	M A I N T A I N A B I L I T Y	V E R I F I A B I L I T Y	E X P A N D A B I L I T Y	I N T E R O P E R A B I L I T Y	P O R T A B I L I T Y	R E U S A B I L I T Y
MONITORING SYS	A	A	A	G	G	A	E	E	G	G	E	
ANALYTIC FWD PR	G	A	E	G	G	A	E	E	G	G	E	
DEVICE KNOWLEDGE	A	A	A	A	G	A	E	E	A	G	E	
SYSTEM												

NOTE FOR GOAL ENTRIES:

E = EXCELLENT
G = GOOD
A = AVERAGE
BLANK OR N/A =
NOT IMPORTANT OR
NOT APPLICABLE

Functions and Quality Factors for the

ENEMY PERFORMANCE ASSESSMENT AID

SOFTWARE QUALITY FACTOR	PERFORMANCE					DESIGN			ADAPTATION			
	E F F I C I E N C Y	I N T E G R I T Y	R E L I A B I L I T Y	S U R V I V A B I L I T Y	U S A B I L I T Y	C O R R E C T I N E S S	M A I N T E N A B I L I T Y	V E R I F I C A B I L I T Y	E X P A N D A B I L I T Y	I N T E R O P E R A B I L I T Y	P O R T A B I L I T Y	R E U S A B I L I T Y
SYSTEM OR SOFTWARE- UNIQUE FUNCTION												
Assist in	A	-	A	-	A	G	G	E	E	E	G	G
Developing H.R.												
Appointing Reccom.												
Provide Rational	A	-	A	-	A	G	A	A	A	G	G	A
+ Building Reports												
for App. Comm.												

NOTE FOR GOAL ENTRIES:

E = EXCELLENT
G = GOOD
A = AVERAGE
BLANK OR N/A =
NOT IMPORTANT OR
NOT APPLICABLE

Functions and Quality Factors for the
Resource Apportionment Aid.

APPENDIX D

SAMPLE QUALITY EVALUATION FORMS

This appendix contains samples of forms used during the study. The following pages contain:

- Metric Element Scoring Form
- Metric Element Evaluation Form
- Worksheet Evaluation Form

The Metric Element Scoring Form was used to capture answers to all metric elements for each worksheet. To accompany the answer, this form also captures the rationale and location from the documentation where the answer was found.

The Metric Element Evaluation Form was used to capture information for use in evaluating each metric element. Areas which were included were applicability, phrasing of question, and measurement effort.

The Worksheet Evaluation Form was used to evaluate how well each criterion was measured by each worksheet. Also, the applicability of the criterion to the life cycle phase was measured.

METRIC ELEMENT SCORING FORM

SYSTEM _____ FUNCTION / CSCI _____

PHASE _____ W.S. # _____ UNIT _____

METRIC ELEMENT ID (E.G. AANNN) _____ EVALUATOR _____

ANSWER TO QUESTION(S)

(List each sub-question separately, e.g. a,b,c₁, c₂, etc.)

RATIONALE FOR ANSWER

(continue on separate page if necessary)

INFORMATION SOURCES USED

(List each source: Document Name, Para. #, Pg. #)

CHECK HERE IF NO INFORMATION WAS FOUND _____

1. _____
2. _____
3. _____
4. _____

METRIC ELEMENT EVALUATION FORM

SYSTEM _____ PHASE _____ W.S. # _____
 METRIC ELEMENT ID [E.G. AAN(NN)] _____ EVALUATOR _____

AREAS OF CONCERN

(Mark all items which apply. Describe specific concerns in 'comments' area, below)

	SOME RESERVATIONS	MAJOR CONCERNS
METRIC ELEMENT DOES <u>NOT</u> APPLY TO:		
1. This CRITERION	_____	_____
2. This lifecycle PHASE	_____	_____
3. This SYSTEM	_____	_____
4. Knowledge Based Systems	_____	_____
PHRASING OF QUESTION:		
5. Question is UNCLEAR or AMBIGUOUS	_____	_____
6. Question is SUBJECTIVE rather than OBJECTIVE	_____	_____
7. Question is biased toward		
A. An operating system	_____	_____
B. An implementation language(s)	_____	_____
C. Specific hardware	_____	_____
D. Command Control Systems	_____	_____
8. Question precludes or is missing important considerations (list in comments below)	_____	_____
MEASUREMENT EFFORT/ACCURACY:		
9. Value is difficult to determine	_____	_____
10. Formula is incorrect	_____	_____
MARK ALL <u>'YES' ANSWERS</u>		
11. Expertise required for this question:		
SYSTEMS ENGINEERING	_____	
SOFTWARE METHODOLOGIES	_____	
OPERATING SYSTEMS	_____	
LANGUAGE(S)	_____	
HARDWARE	_____	
12. Question is automatable	_____	

METRIC ELEMENT EVALUATION FORM

W.S. # _____ METRIC ELEMENT ID (E.G. AAN(NN)) _____ EVALUATOR _____

OTHER CONCERNS:

(List additional issues with the metric element)

COMMENTS:

(Include Item No. for Each Comment. Continue on separate page if necessary.)

[illegible]

RECOMMENDED ACTIONS

(Mark all that apply)

DELETE QUESTION _____ REPLACE QUESTION _____

SPLIT INTO SUB-QUESTIONS _____ **FIX TYPO** _____

WORKSHEET EVALUATION FORM

SYSTEM _____ PHASE _____ W.S. # _____ EVALUATOR _____

CRITERIA APPLICABILITY & COMPLETENESS

Describe all NO and POOR answers in comments area, below

CRITERIA	IS CRITERION APPLICABLE TO THIS PHASE?		HOW THOROUGHLY IS CRITERION MEASURED?		
	YES	NO	POOR	ADEQUATE	GOOD
1. ACCURACY	—	—	—	—	—
2. ANOMALY MANAGEMENT	—	—	—	—	—
3. APPLICATION INDEPENDENCE	—	—	—	—	—
4. AUGMENTABILITY	—	—	—	—	—
5. AUTONOMY	—	—	—	—	—
6. COMMONALITY	—	—	—	—	—
7. COMPLETENESS	—	—	—	—	—
8. CONSISTENCY	—	—	—	—	—
9. DISTRIBUTEDNESS	—	—	—	—	—
10. DOCUMENT ACCESSIBILITY	—	—	—	—	—
11. EFFECTIVENESS - COMMUNICATION	—	—	—	—	—
12. EFFECTIVENESS - PROCESSING	—	—	—	—	—
13. EFFECTIVENESS - STORAGE	—	—	—	—	—
14. FUNCTIONAL OVERLAP	—	—	—	—	—
15. FUNCTIONAL SCOPE	—	—	—	—	—
16. GENERALITY	—	—	—	—	—
17. INDEPENDENCE	—	—	—	—	—
18. MODULARITY	—	—	—	—	—
19. OPERABILITY	—	—	—	—	—
20. RECONFIGURABILITY	—	—	—	—	—
21. SELF-DESCRIPTIVENESS	—	—	—	—	—
22. SIMPLICITY	—	—	—	—	—
23. SYSTEM ACCESSIBILITY	—	—	—	—	—
24. SYSTEM CLARITY	—	—	—	—	—
25. SYSTEM COMPATIBILITY	—	—	—	—	—
26. TRACEABILITY	—	—	—	—	—
27. TRAINING	—	—	—	—	—
28. VIRTUALITY	—	—	—	—	—
29. VISIBILITY	—	—	—	—	—

WORKSHEET EVALUATION FORM

SYSTEM _____ **PHASE** _____ **U.S. °** _____ **EVALUATOR** _____

COMMENTS:

(Include Criterion No. for Each Comment. Continue on separate page if necessary.)

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There is no handwriting or other markings on the paper.

OTHER CONCERNS:

(List areas not addressed by this worksheet, add'l metrics, metric elements, etc.)

APPENDIX E

SOFTWARE QUALITY REQUIREMENTS REPORT:

SAMPLE TABLE OF CONTENTS

This appendix contains a sample Table of Contents for the RAA SQRR.

1.0	Introduction
1.1	Purpose
1.2	Organization
1.3	Decision Aid Description
1.4	Glossary of Terms
2.0	References
2.1	Government Documents
2.2	SBSDA Documents
3.0	Software Quality Specification
3.1	Specification Source Materials
3.1.1	Software Quality Specification Guidebook
3.1.2	RAA Documentation
3.1.3	Quality Requirements Surveys
3.2	Quality Specification
3.2.1	Quality Specification for the Apportionment Recommendations Function
3.2.1.1	Quality Factor Selection
3.2.1.1.1	Initial Factor Goal Determination
3.2.1.1.2	Final Factor Goal Determination
3.2.1.1.2.1	Accounting for Positive and Negative Interrelationships
3.2.1.1.2.2	Establishing Numeric Ranges for Quality Goals
3.2.1.2	Criteria Selection and Weighting
3.2.1.2.1	Initial Criteria Selection and Weighting
3.2.1.2.2	Final Criteria Weighting
3.2.1.3	Metric and Metric Element Selection
3.2.2	Quality Specification of the Provide Rational Function
3.2.2.1	Quality Factor Selection
3.2.2.1.1	Initial Factor Goal Determination
3.2.2.1.2	Final Factor Goal Determination
3.2.2.1.2.1	Accounting for Positive and Negative Interrelationships
3.2.2.1.2.2	Establishing Numeric Ranges for Quality Goals
3.2.2.2	Criteria Selection and Weighting
3.2.2.2.1	Initial Criteria Selection and Weighting
3.2.2.2.2	Final Criteria Weighting
3.2.2.3	Metric and Metric Element Selection

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C³I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C³I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.

END

DATE

FILMED

6-1988

DTIC