

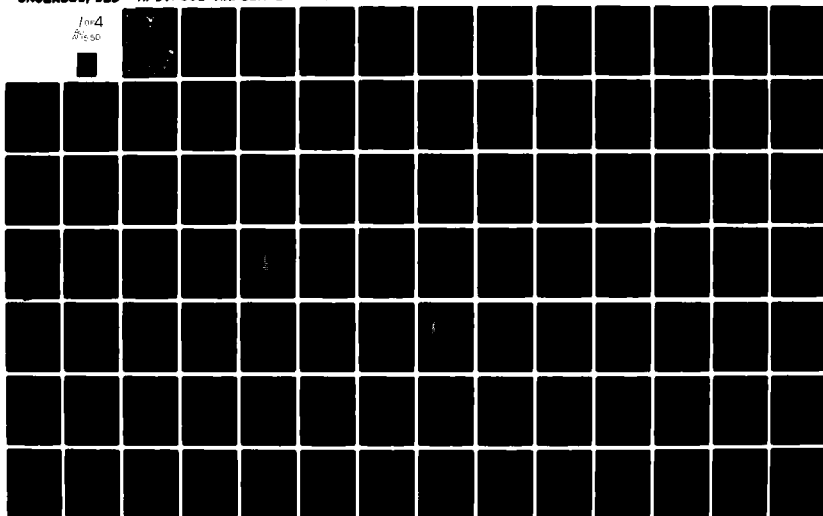
AD-A115 501

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/8 9/2
SOFTWARE QUALITY METRICS: A SOFTWARE MANAGEMENT MONITORING METH--ETC(U)
MAR 82 S J JARZOMBK
AFIT/GCS/HA/62H-1

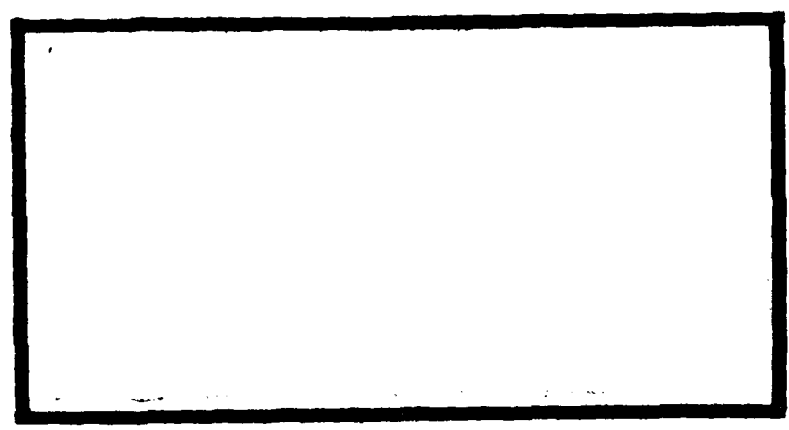
UNCLASSIFIED

NL

1 of 4
8/1/80



AD A115501



DTIC FILE COPY

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DTIC
ELECTE
JUN 14 1982
H

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

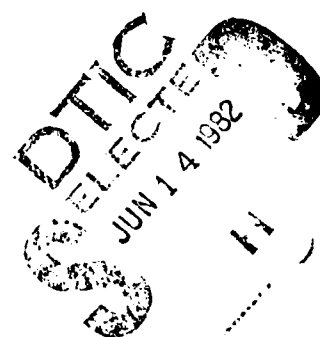
82 06 14 157

2

SOFTWARE QUALITY METRICS: A SOFTWARE
MANAGEMENT MONITORING METHOD FOR
AIR FORCE LOGISTICS COMMAND IN ITS
SOFTWARE QUALITY ASSURANCE PROGRAM
FOR THE QUANTITATIVE ASSESSMENT OF
THE SYSTEM DEVELOPMENT LIFE CYCLE
UNDER CONFIGURATION MANAGEMENT

THESIS

AFIT/GCS/MA/82M-1 Stanley J. Jarzombek, Jr.
2nd Lt USAF



SOFTWARE QUALITY METRICS:

A SOFTWARE MANAGEMENT MONITORING METHOD FOR
AIR FORCE LOGISTICS COMMAND IN ITS SOFTWARE QUALITY
ASSURANCE PROGRAM FOR THE QUANTITATIVE ASSESSMENT OF
THE SYSTEM DEVELOPMENT LIFE CYCLE
UNDER CONFIGURATION MANAGEMENT

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Stanley J. Jarzombek, Jr., B.A., B.B.A.
2nd Lt USAF
Graduate Computer Information Systems

March 1982

Accession For	
NTIS GRA&I	
DTIC TAB	
Unannounced	
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	



Approved for public release; distribution unlimited.

PREFACE

I wish to express my sincere appreciation to my thesis advisor, Dr. Daniel E. Reynolds, for his guidance, interest, and motivation throughout this thesis effort.

I wish to thank Ray Ruby for motivating my initial interest in software quality assurance and for providing comments to this thesis.

Thanks go to Major Ross, who contributed comments and input to this thesis effort.

A note of appreciation goes to the personnel in AFLC/LM, especially Richard Woodward, who provided valuable information about AFLC's software development procedures.

A special note of acknowledgement is due to Dave Markham and James McCall for providing recent information concerning the development of software quality metrics.

A very special expression of appreciation is due my wife, PJ, for her patience, understanding, and support, which helped me accomplish this task.

Final thanks go to Sheila Finch for her dedication and competence in the typing of this thesis.

CONTENTS

	Page
PREFACE.	ii
LIST OF FIGURES.	v
LIST OF TABLES	vi
ABSTRACT	vii
 I. INTRODUCTION: BACKGROUND INFORMATION.	 1
Software Life Cycle Implications	2
AFLC's Software Development Efforts.	5
Failures of Current Development Efforts.	7
Current SQA Efforts.	9
Measuring Software Quality	12
Justification for SQA.	13
Conceptualization of the Decision-Making Process.	14
Front-End Planning Through Configuration Management	16
A Framework for Software Quality Assessment.	17
Background Abstract.	20
Problem Statement.	21
Research Objectives.	21
Scope of Research.	22
 II. SOFTWARE QUALITY METRICS: AN OVERVIEW.	 24
Quality Measurement in Perspective	24
Categories of Metrics.	25
Complementary Use of Metrics with SQA.	25
Comparison of Metrics, Inspections, and Walk-Throughs.	28
Identifying Software Quality Requirements.	30
Techniques for Applying Metrics.	42
 III. RESEARCH METHODOLOGY.	 44
Establishing the Framework	44
Model of Development Decision Process.	46
Metric Application	49
Metric Integration	50
 IV. SQA TOOLS AND TECHNIQUES: A SURVEY OF DOD AND INDUSTRY	 51
Techniques, Tools, and Methodologies	53
Toolsmithing	57
Automated Measurement Services	64
Report Generation Services	68
Chapter Summary.	73

CONTENTS

	Page
V. SDLC DECISION-MAKING MODEL: A MAPPING TO SQM . .	75
The Software Development Life Cycle.	75
Modeling the Decisioning Process	77
Imposing Metric Framework on the Model	81
Monitoring the Development Effort.	86
A Conceptual Walk-Through.	88
Metric Application Throughout the SDLC	92
Chapter Summary.	107
VI. SQA INFORMATION REQUIREMENTS: A CHECKLIST. . . .	109
Documentation Requirements	109
Application for Checklists with SQM.	111
VII. THESIS RECOMMENDATIONS: GUIDELINE FOR A SQA PROGRAM.	120
Research Summary	120
Research Recommendations	122
REFERENCES USED.	126
APPENDIX A: METRIC WORKSHEETS	138
APPENDIX B: WORKSHEET REQUIREMENTS.	153
APPENDIX C: EXPLANATION OF METRICS.	160
APPENDIX D: METRIC ALGORITHMS	198
APPENDIX E: DEFINITION OF QUALITY ATTRIBUTES.	208
APPENDIX F: SOFTWARE SYSTEM DEVELOPMENT LIFE CYCLE FOR AFLC/LM.	223
APPENDIX G: GLOSSARY OF TERMS FOR SOFTWARE CONFIGURATION MANAGEMENT.	268
APPENDIX H: GLOSSARY FOR SQA TOOLS AND TECHNIQUES . .	283
APPENDIX I: TOOL SURVEY	295
APPENDIX J: PROCEDURE FOR ASSESSING SOFTWARE QUALITY	307
APPENDIX K: SAMPLE OF AMT REPORTS	317

LIST OF FIGURES

Figure		Page
1	Software Development Life Cycle (SDLC). . .	3
2	A Software Development Problem.	4
3	Configuration Management for AFLC	10
4	The Software Development-Environment Interaction System.	15
5	Software Quality Framework.	19
6	Application of the Metric Worksheets. . . .	41
7	AFLC's Software System Development Life Cycle.	76
8	Software System Development-Environment Interaction System.	80
9	Defining the User's System Requirements . .	82
10	SQM Applied to the Decision-Making Process.	89
11	Requirements Analysis	93
12	Design Phase.	97
13	Programming and Checkout.	101
14	Test and Integration.	105
15	The Mapping of SQM to the SDLC.	108
16	ADS Development	224
17	Matrix of Software Tools Having Metric Applicability	300
18	Tool Usage.	306
19	Normalization Function for Flexibility During Design	314
20	Determination of Level of Confidence. . . .	315

LIST OF TABLES

Table		Page
I	How Software Metrics Complement Quality Assurance.	26
II	Comparison of Key Properties of Inspections, Walk-Throughs and Metrics . . .	29
III	Software Quality Requirements Survey Form. .	32
IV	System Characteristics and Related Quality Factors.	33
V	Impact of Not Specifying or Measuring Software Quality Factors	34
VI	Relationships Between Software Quality Factors.	35
VII	Some Tradeoffs Between Software Quality Characteristics.	36
VIII	Software Criteria and Related Quality Factors.	38
IX	Software Attributes Identified in the Literature	52
X	SQA Tools and Techniques	55
XI	Supporting Techniques for SQA Functions. . .	56
XII	Technique Effectiveness in Assessing Quality Properties	58
XIII	Supporting Tools for SQA Functions	59
XIV	Tool Effectiveness in Assessing Quality Properties	60
XV	Factors Impacting Tool and Technique Selection.	62
XVI	Minimum Set of Tools and Techniques.	63
XVII	Automated Metric Data.	66
XVIII	Support to Personnel	72
XIX	Quality Factor Ratings	85
XX	Definitions Relating to Reliability and Maintainability.	87
XXI	Normalization Functions.	312

ABSTRACT

Software Quality Assurance (SQA) is recognized as an essential function needed to monitor the software system development life cycle (SDLC). The inability to objectively assess the quality of the software system, as it is being developed, has caused implementation problems, cost and schedule overruns, and the delivery of end-products that are difficult to maintain and often fail to satisfy user requirements.

The framework established for Software Quality Metrics (SQM) provides goal-directed system specifications and the ability to quantitatively assess the quality of the system under development. The Automated Measurement Tool (AMT), which operationalizes the application of SQM, functions as the core of a Decision Support System, providing quantitative measures and various levels of reports.

An analysis of current SQA aids enabled the recommendation of a minimum set of tools and techniques to be used by the SQA program for monitoring the SDLC.

The SDLC has been envisioned as an iterative process controlled by management. Recognizing the functional impact of specific information as the key to objectively

monitoring and controlling the software system development, the decision-making model was conceptualized as three subsystems within each phase of the SDLC: scanning (afferent), organizing (intelligence), and decision (efferent). Because the SQM concepts are based on the availability of key information, the discussion of the model pinpointed the timing for the appropriate documents that should contain specific information which is needed to assess the software quality. Moreover, the use of checklists by system developers highlights a prescriptive method of goal-directed development.

In all, the thesis provides justification for using Software Quality Metrics by reviewing the need and demonstrating how the SQM concepts can now be used by AFLC/LM.

CHAPTER I

INTRODUCTION: BACKGROUND INFORMATION

In recent years there has been an increased emphasis to establish Quality Assurance (QA) programs at all levels within the Department of Defense. DoD Directive 4155.1 defined Quality Assurance as a planned and systematic pattern of all actions necessary to provide adequate confidence that material, data, supplies, and services conform to established technical requirements and achieve satisfactory performance (Ref 37).

The use of MIL-S-52779A, Software Quality Assurance Program Requirements, in DoD contracts (requiring government contractors to establish a software QA function), has generated an increased awareness of the need for QA in the development of software; yet many DoD organizations have been unable to internally establish Software Quality Assurance (SQA) programs (Ref 52). The Air Force Logistics Command (AFLC) has recognized the need for a SQA program mainly because of the cost of producing and maintaining its wide spectrum of conventional software systems: Management Information Systems (MIS) for Plans/Programs/Budget, Data Base systems, Inventory systems, Accounting systems, and Automated Logistics

Maintenance systems (Ref 38). AFLC has recognized that "maintenance costs for software range from 50-75% of total system costs, thus, the command position is to design and develop software that is easy to maintain in order to reduce operational phase costs" (Ref 89).

Organizationally, AFLC has six main users of conventional software systems: Procurement & Management (PM), Personnel (MP), Logistics Operations (LO), Maintenance (MA), Comptroller (AC), and Logistics Management (LM). AFLC/LM is responsible for software development within AFLC. It provides the Project Management (LMX), Configuration Management and Project Engineering (LME), Technical Support (LMT), ADP Resources (LMD), Computer Operations (LMO), and Training (LMR) for the development of software systems. Once the system is operational, it is "turned-over" to the user of the system, i.e., AFLC/LO (Ref 48).

Software Life Cycle Implications

AFLC recognizes that by implementing a SQA program it will be enhancing the software development by imposing a QA discipline on the software life cycle. The Software Development Life Cycle (SDLC), as described in Dod Standard 7935.1-S, provides for the structured implementation of software from conceptualization through operations (Ref 47).

Represented in Figure 1, the SDLC begins with the conceptualization or Requirements Analysis. This phase

determines "what" the software system is to do. Next the Design phase describes "how" the requirements will be met. Following this is the Programming (Coding) and Checkout phase, which entails the greatest amount of manpower and funds. While the test phase concludes the normal development cycle, one should realize that with software the development continues in the Operations and Maintenance phase because of debugging and changes in the user requirements (Ref 11).

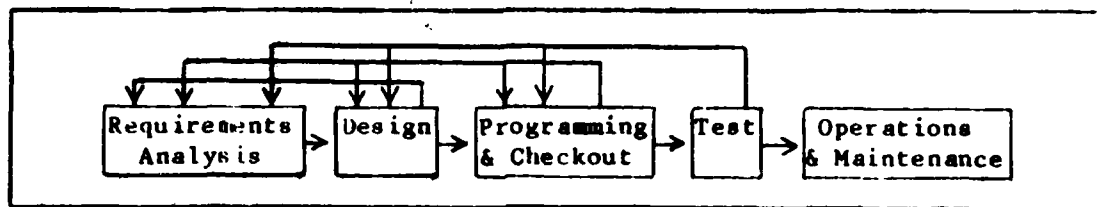


Figure 1.
Software Development Life Cycle (SDLC)

The arrows in Figure 1 indicate the possible paths of software development. They demonstrate that the development phases are iterative in nature. The iterations show that the identification of problems causes the development effort to "fall-back" on to earlier phases in order to correct the problem. One should be aware that there are associated costs for such problems or errors. Figure 2 reveals that the earlier an error is made, the later it will be detected in the SDLC (Ref 14). Thus, it costs more to correct errors made during the requirements analysis than it does for errors made in coding because later detection of an error equates to the use of

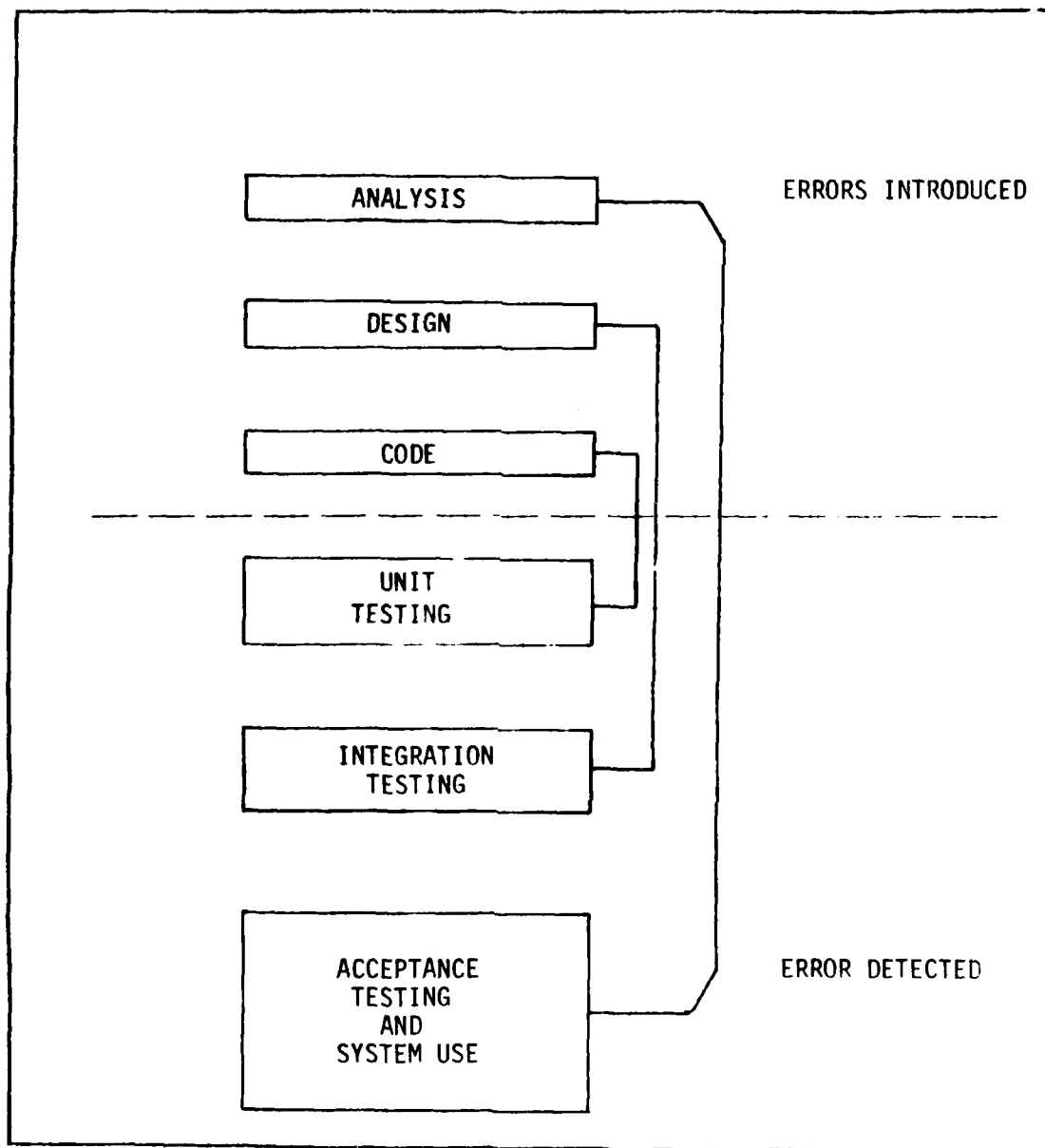


Figure 2.
A Software Development Problem

additional time and resources. Furthermore, extensive analysis of error data by TRW reveals that most errors are design and requirements errors, as opposed to coding errors (Ref 78).

With these facts in mind, it would appear logical to assume that any tool that can improve the quality of the requirements analysis would have a significant impact on the overall development cost. Moreover, it follows that there would be a genuine attempt to find ways of improving quality, hence reducing the overall costs by saving time and resources. However, as of this date, AFLC uses no quality measurement tools during the requirements analysis or design phase. The consequence of this vacuum created by the lack of up-front measurement has been the delivery of software systems that must be modified in order to meet user needs (Ref 38).

AFLC's Software Development Efforts

Efforts to develop quality assurance programs for software systems at AFLC began in 1974 with the Advance Logistics System (ALS). This was designed to integrate the logistic software systems. However, because ALS was late in being developed at a cost that had far exceeded its budgeted amount, AFLC was forced to firm up the software management discipline.

The likely candidate to help accomplish this task was Configuration Management (CM) because it had been used for years in the development of weapons systems and, as a

result, people thought that it could be transferred to software development. A "get-well" plan was implemented which brought in CM people from the weapons SPOs (System Program Office) to create an ALS SPO (Ref 42).

In July 1975 the new ALS SPO developed AFLCR 300-3, a document which at that time was viewed as being "on-the-cutting-edge" for software configuration management. Unfortunately, the new regulation was developed too late to improve the development management of ALS; so in December 1975 Congress killed ALS. As a result, the development of software configuration management was halted for several years after that (Ref 42).

In January 1978 AFR 300-15 (built on AFLCR 300-3 and AFDSDC 300-8) was published (Ref 47). Although this document includes a one page chapter entitled "Quality Assurance", it falls short of insuring a quality software product. While the regulation says that quality assurance will exist, it fails to specify what such a program should involve. This is evidenced by the fact that AFR 300-15, like the other regulations that govern software development, makes no attempt to define or measure quality.

Within AFLC/LM, quality assurance is supposed to be a function within Configuration Management, but in reality, "CM fails to provide software QA." This sentiment of "having QA" contributes to a lack of urgency. Moreover, because the CM people are more hardware oriented, they lack the skills for software QA (Ref 42). In other words,

weapons systems (hardware) CM has not readily transferred to software development.

Failures of Current Development Efforts

Another factor which inhibits an effective Quality Assurance effort under the current AFLC/LM structure is the dependence between Configuration Management and Program Management (Ref 42). Ray Rubey from SOFTEC calls this "Cognitive Dissonance" because individuals are asking people to report where they went wrong when they truly believe that they did a good job (Ref 44). In other words, SQA should be an independent effort by a functionally separate team (Ref 52). Moreover, a formal SQA program would indicate commitment by management to the requirements for a quality operation, and it would make SQA a visible and identifiable function (Ref 38).

In November 1981, at a symposium at AFIT, Ray Rubey identified some of the reasons why SQA can fail (Ref 44). Four of the five reasons would seem to directly apply to AFLC/LM in software systems development effort:

1. QA is not included in the budget. Although a "flavor" of QA is expected from CM, no dollars are directly allocated for SQA (Refs 38, 42). Quality Assurance costs include QA personnel salaries, administrative costs, computer time, etc. If resources or programs are not budgeted at the beginning of a project, it is difficult to expect that funds will later be found for them.

2. Inexperienced or incompetent people are on QA

teams. AFLC/LM personnel admitted that CM people are not trained in SQA (Ref 42).

3. QA does not start when the development starts. Currently, even CM reviews do not occur until after the requirements are specified, and there is no SQA effort in AFLC/LM (Refs 38, 89).

4. QA and system objectives are unknown. CM does not even address the basic question -- What do we mean by quality? The objectives and quality factors are not identified (Ref 38). AFR 300-15 does state that "QA policies and procedures must be set up which tell how to conduct configuration audits, and how to make sure that the CPCI meets ADP standards..." Yet if the policies, procedures, and standards fail to mention anything about measuring quality (which is the case), then simply complying with them also fails to insure quality software. Richard Woodward in AFLC/LM, the sponsor of this research, has indicated that CM fails to provide standards for acceptance and that under CM there is no way to determine the quality of the software system. Clearly, "AFLC currently has no command level Quality Assurance program for software" (Ref 38).

Currently, there is a lack of clear evidence that SQA is a reality, despite the attempt of AFR 300-15 to claim QA is in place. Indeed, defenders can point to phrases that imply standards must be followed, but which standards are to be followed? In addition to shortcomings already

pointed out, AFR 300-15 provides no criteria for determining which standards, if any, should be used. Personnel in AFLC/LM noted that there is no directive that sets one standard for software development for AFLC/LM -- there are several different standards, and none are quantified (Ref 48).

Although Configuration Management has failed to provide the standards for acceptance, and hence failed to quantify quality, it does provide the framework on which a SQA program can be built because it establishes the structure necessary to enforce compliance with procedures. Using DODD 7935.1-S, AFR 300-12, and AFR 300-15, AFLC/LME described the SDLC, as depicted in Figure 3, to provide a structured model for software development (Refs 46, 47, 85). Because this structure governs the software development environment for AFLC/LM, it is presented in more detail in Appendix F which provides a listing of all deliverable documents for each milestone (review) in the SDLC (Ref 89). A glossary of terms for the software Configuration Management follows in Appendix G which provides a description of each review and a definition for CM terms.

Current SQA Efforts

With a development structure firmly in place, AFLC/LM has begun to define requirements for a Software Quality Assurance Program. A formal SQA program would indicate commitment by management to the requirements for a quality

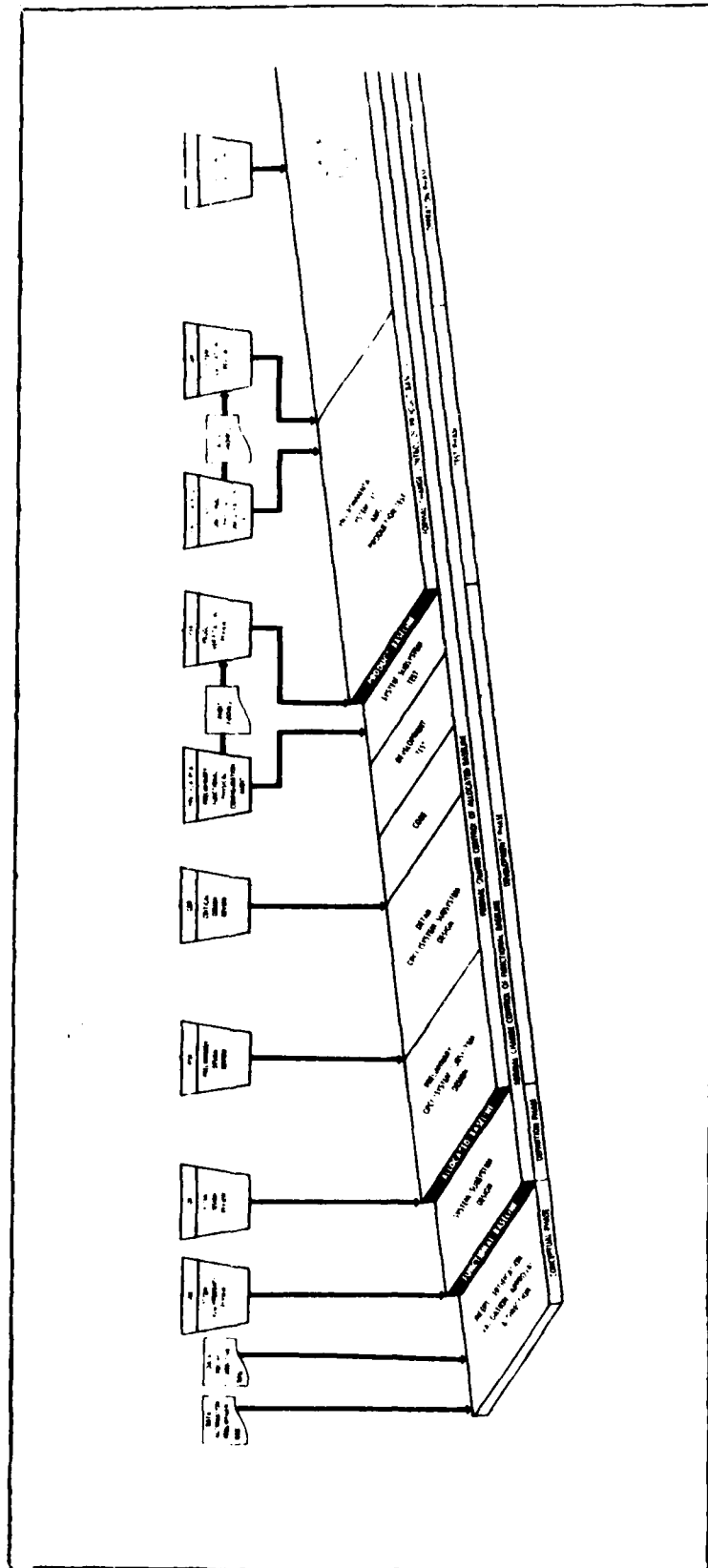


Figure 3. Configuration Management for AFLC

operation. It would also make SQA a visible/identifiable function to management and to the development staff (Ref 38). Initially, three alternatives to the SQA program were proposed by AFLC/LM.

The first alternative, called the "Broad Program," would have used a quality audit approach applied to all systems at varying levels. At the higher level would be a consideration of compliance with DoD/Air Force directives and a checking for effective operational interface. The lower level would get more into specifics, evaluating performance against designated standards and requirements.

The second alternative, labeled "Main Projects," would have implemented a quality assurance program that would subject major projects to tailored evaluations or considerations. Evaluations then could be made that were specific to each particular system. Systems not classified as major would be subjected to a less vigorous review or audit. Although this alternative would be the least costly of the three alternatives, it would probably also be the least effective.

The third alternative, called the "Cradle to Grave" quality program implies QA evaluations, such as checklists and standards, throughout the life cycle of all systems. It is designed to involve users of the system, require documenting of all significant actions, provide error tracking and auditing capabilities, and include quality checks for systems interfacing with the basic system. The

plan states that documentation and related actions for each phase must be evaluated and accepted by a QA task group before proceeding to the next phase and before releasing the system for implementation.

Because the third alternative includes projects of all sizes and would operate throughout the total life cycle of software development, it will be adopted as AFLC's Software Quality Assurance Program (Ref 38).

One of the basic requirements for the success of this SQA program is that a "carefully prepared acceptance checklist should be developed." (Ref 38) To implement such a checklist, tools are needed to quantify the quality of the software under development. Fortunately, these tools are now available through certain sectors of industry.

Measuring Software Quality

Recognizing the importance of such tools, DoD has recently started funding efforts which are attempting to adapt industrial research to military requirements. This research has resulted in the development and evaluation of a number of metrics purporting to measure various qualitative attributes of software. These metrics, once established, can provide quantifiable measures for software quality (Ref 23).

The metric measurements are derived from implied standards which are accepted as being necessary for quality software. Basically, the metrics are a check on the development procedures to determine if certain standards

were incorporated (regardless if they were stated or not).

Justification for SQA

The need for software quality assurance has been demonstrated in a Governmental Accounting Office (GAO) report which reviewed nine federal software projects totaling \$6.75 million. This total included:

- \$3.20 M of software that was delivered, but never used,
- \$1.95 M of software that was paid for, but not delivered,
- \$1.30 M of software that was used temporarily, then abandoned,
- \$.20 M of software that was used only after changes were made,
- \$.10 M of software that was used as it was delivered (Ref 40).

In other words, less than 1.5% of the total funds spent on software actually produced software Products that could be used by the user as delivered. The report suggested that if measurable standards were applied during the systems development life cycle, then the user of the software could be better assured of receiving a usable software system.

In recent years, symptoms indicating the existence of an inadequate quality assurance program have been encountered in the development of large scale systems. These symptoms include: cost and schedule overruns, poor performance of the systems once they are delivered, high maintenance costs, lack of reliability, and a high degree of system sensitivity to changes in requirements. Examples of these situations have been well documented in the literature (Refs 1,2,3,4,5,6,7). In short, government and industry sources have identified the enforcement of

quantifiable standards throughout the system development life cycle as one key to delivering quality software systems.

Conceptualization of the Decision-Making Process

In order to understand why current software systems development efforts have failed to enforce measurable standards, one has to have an adequate conceptualization of the decision-making process which is involved in creating a complete software system. In Figure 4, the software development process is represented as an open system with relationships between its own internal organization and the environment (Ref 49).

During the software system development effort for AFLC/LM, Configuration Management provides the structure for the three subsystems: Scanning, Organizing, and Decision. Conceptually, the actions of AFLC's system development organization, which are the outputs of the decision subsystem, should not be based on the outputs of the scanning subsystem, but rather upon the outputs of an intelligence (organizing) subsystem which would act as an evaluator of the receptor or scanning subsystem. Data received by the scanning (receptor) system needs to be eventually fed into the decision system where it can be utilized for problem-solving purposes (Ref 49).

It is the data which is currently collected that makes the eventual decisions ineffective. The current scanning (receptor) data is the result of checking for compliance

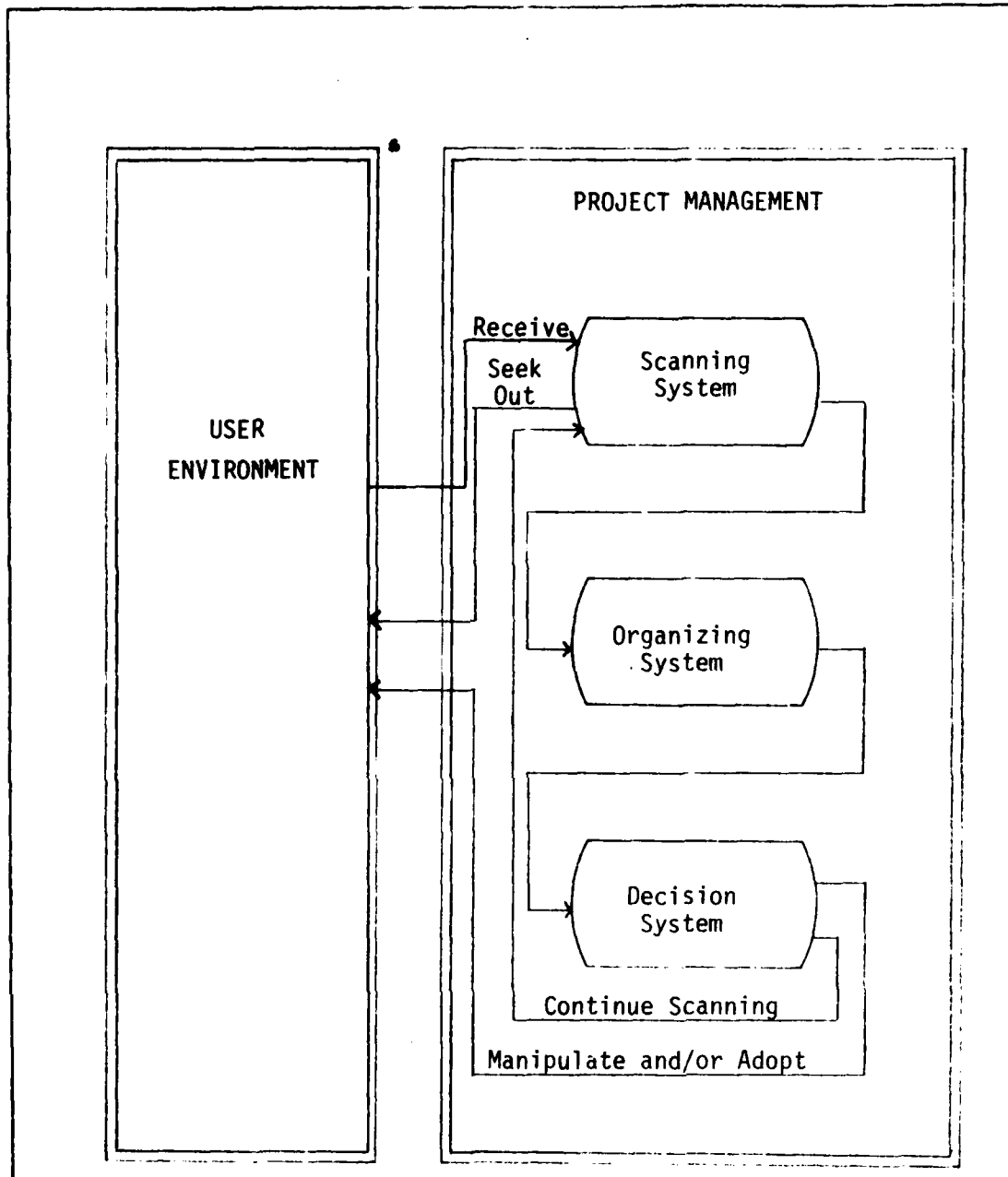


Figure 4.

The Software Development-Environment Interaction System

with outputs, i.e., to check if source documents are being produced to monitor the software development; yet the needed information should check the quality of that documentation. AFLC personnel noted that although CM insures that the documents are produced, that there is no check to insure the quality of the document (Ref 48).

A software QA program can provide the tools, such as Software Metrics, which would indicate the quality of the documentation being produced. This is precisely why the data collected by such a SQA program would be inherently different from that collected under CM. Such metric data would indicate the quality of the documentation rather than just its existence. By using Software Metrics the organizing (intelligence) subsystem would be able to relate the data into meaningful measures which are then passed on to the decision subsystem. This model of the decision-making process of the software development effort will be used in Chapter 5 to demonstrate how software metrics can be applied by AFLC/LM.

Front-End Planning Through Configuration Management

Additionally, Configuration Management provides a planning structure to the development effort. In the development of a software system, AFLC needs to specify a system's requirements, and then be able to determine whether those system requirements are being satisfied as the software system evolves. The parameters of the specification center around the technical definition of the

application and the software role within the overall system. CM aids in this front-end planning and could greatly increase the software management effort if it incorporated a definition of quality and the system objectives (Ref 42).

While the application functions, cost, and schedule aspects of development can be objectively defined, measured, and assessed throughout the development of the system, the quality desired has historically been definable only in subjective terms. This occurs because there are no quantifiable criteria against which to judge the quality of the software until the system is under operational conditions (Ref 38). Current scheduling methods, such as PERT/CPM or GANTT charts tell nothing about the quality of the system. Rather, they provide information about the state of existence, i.e., is the software completed by a certain time within a specified cost (Refs 43, 45).

A Framework for Software Quality Assessment

Fortunately, there is a framework for software quality metrics that can provide the measurement tools needed to quantitatively assess software quality. This framework has evolved over a number of years. Boehm traces some of the previous work contributing to this evolution (Ref 19). The work dates back to a paper by Ray Rubey and R. Hartwick in 1968 which first introduced the concept of software metrics (Ref 20). Later studies established a more formal conceptual framework (Refs 21, 22).

The catalyst that set off this attempt to quantify attributes of software was the introduction of more formal and structured software design, implementation, and review techniques -- a framework which was conducive to the quantitative measurement of software quality (Ref 31). Figure 5 illustrates the hierarchical nature of this framework.

At the highest level, the major aspects (factors) of software quality are identified. The user is the program manager or the customer of the software system. The user requires a defined set of factors in order to identify what qualities are desired in the software system being developed. To satisfy this use, the definitions of the factors must lend themselves to quantification (measurement) that is meaningful to users.

The approach taken to satisfy these requirements is to evaluate how a program manager views the end product of a software development. The orientations or viewpoints identified relate to life cycle activities involving the software product (Ref 18).

Underlying these user-oriented quality factors is a set of attributes (criterion) which, if present in the software, provide the characteristics represented by the factors. For each factor then, a set of criteria has been established and defined (Ref 16).

A key point in the approach is that the measurements are to be taken during the development effort. These

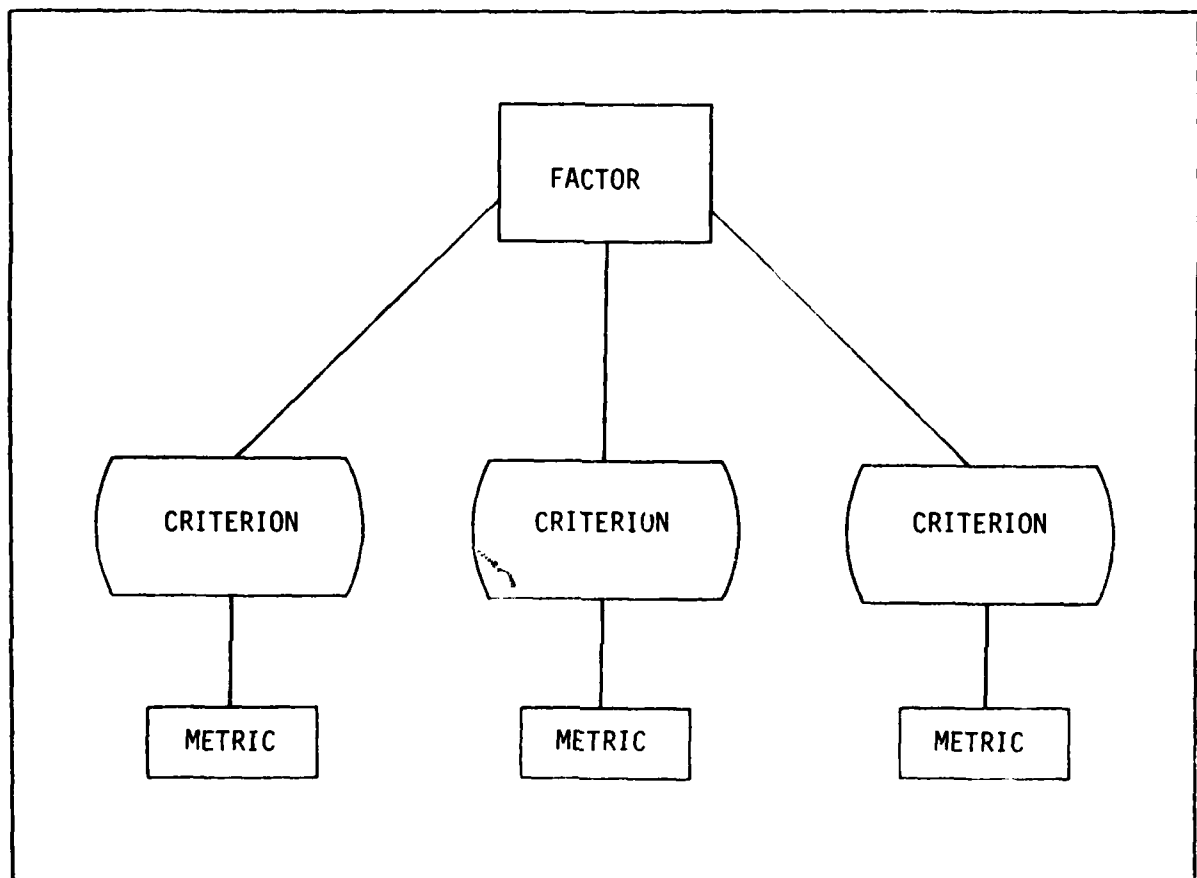


Figure 5.
Software Quality Framework

measurements are not post-implementation assessments of software quality. Their purpose is to provide an indication of the progression toward a desired level of quality. The set of attributes, or criteria, established for each quality factor then represents attributes which can be measured during the software development (Ref 17).

The framework provides a mechanism for a project manager to identify what qualities are important. These qualities are attributes of the software in addition to its functional correctness and performance which have life cycle implications (Ref 28). Such factors as reliability and maintainability have been shown in recent years to have significant life cycle cost impacts (Ref 24).

Background Abstract

Because of the significant cost and schedule overruns, DoD has tried to place an emphasis on QA programs through Military Standards, yet AFLC has been unable to incorporate these in the development of its conventional software systems. The quantitative nature of software metrics provides the measurement tool needed in a SQA program to determine the qualitative attributes of software. By having a tool by which to objectively measure software quality, AFLC/LM will be able to enhance the controllability of software development effort by quantifying the information used in SDLC decision-making.

In summary, AFLC currently has no quantifiable measures for software quality. Even though they have tried and are

continuing to try to define programs to develop quality software, AFLC/LM personnel have not been able to implement the measurement tools that can provide quantifiable quality (Ref 38).

Problem Statement

AFLC needs a software quality assurance (SQA) program that can quantitatively measure the quality of computer software during all stages of its development. Current attempts to establish such a program have been frustrated by program management's inability to define the critical information needed to operationalize such a program.

Research Objectives

The overall objective of this study is to provide recommended guidelines to AFLC/LM concerning ways to objectively specify and quantitatively measure the desired amount of quality in a computer software system during all stages of its development. To accomplish this, the following four goals had to be attained:

1. Determine which measurement tools and techniques should be used by AFLC/LM when it establishes its Software Quality Assurance (SQA) program.

2. Develop a model of the decision-making process of the software system development effort that incorporates the application of Software Quality Metrics and which illustrates the feasibility of implementing a quantitatively oriented SQA program at AFLC/LM.

3. Demonstrate how the SQM concepts can be applied to existing systems at AFLC by developing checklists for system developers.

4. Use the information generated from this research to help AFLC/LM formulate guidelines concerning ways that Software Quality Metrics can be integrated into a command-wide SQA program.

Scope of Research

This research effort concentrated on recommending guidelines for a system to control and monitor the software development process within AFLC.

An attempt has been made to identify the quality metrics and information requirements of an effective software management program. Since reliability and maintainability were identified as the two most critical factors in establishing a SQA program for AFLC (because AFLC must rely on the software systems for several years), this thesis indicates how a software system can be measured to insure reliability and maintainability. It is anticipated that by enhancing AFLC's ability to measure the reliability and maintainability attributes of software quality that a marked improvement in the control of software development will occur.

AFLC/LM, the sponsor of this research effort, has been the central point for gathering data about AFLC's current and past attempts at quality assurance. Data used in discussing the metrics was obtained from the Rome Air

Development Center (RADC) at Griffis AFB, NY. The data was taken from USAF systems that are similar to those at AFLC.

Additionally, the metric worksheets have been applied to G072A, a subsystem of the Maintenance Management Systems Improvement Project (MMSIP) to determine the changes needed by AFLC to be able to use the software metrics.

Because most of AFLC's software development is a product of "in-house" projects, and because AFLC/LM is the OPR for conventional computer systems (not embedded systems), this research will provide recommendations to AFLC/LM.

Chapter II provides an overview of Software Quality Metrics which can provide the quantification needed to measure standards in a SQA program.

CHAPTER II

SOFTWARE QUALITY METRICS: AN OVERVIEW

Quality Measurement in Perspective

The evolution during the past decade of modern programming practices, structured development techniques and methodologies, and requirements for effective documentation, has increased the feasibility of effective measurement of software quality (Ref 16).

However, before the potential of measurement techniques could be realized, a framework or model of software quality had to be constructed. An established model, which at one level provides a user or management-oriented view of quality, is used to establish software quality requirements for a specific application (Ref 17).

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

Categories of Metrics

The metrics can be classified according to three categories: anomaly-detecting, predictive, and acceptance.

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

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

Acceptance metrics are measurements that are applied to the end product to assess the final compliance with requirements. Tests are a form of acceptance-type measurements (Ref 23).

Complementary Use of Metrics with SQA

The measurement concepts of software metrics complement current Quality Assurance and testing practices. They are not a replacement for any current techniques utilized in normal QA programs. Table I summarizes how software metrics complement QA activities by mapping the impacts of software quality metrics onto functions of a SQA program (Ref 16). For example, a major objective of QA is

Table I
How Software Metrics Complement Quality Assurance

QUALITY ASSURANCE PROGRAM REQUIREMENTS	IMPACT OF SOFTWARE QUALITY METRIC CONCEPTS
- Assures conformance with requirements	Expands software requirements
- Identify software deficiencies	Expands deficiency detection with some metrics designed specially for anomaly-detection
- Provide configuration management	Provides measures to determine the adequacy of standards in use by quantifying the quality of products which comply with current procedures
- Conduct tests	Assists in evaluation of other qualities
- Provide library controls	Can provide measurements to determine the adequacy of standards in use
- Review computer program design	Provides metrics that can predict end-product capabilities
- Assure software documentation requirement conformance	Provides metrics that assist in evaluation of documentation as well as code
- Conduct reviews and audits	Provides procedures for applying metrics in the form of worksheets; thus, formalizing the inspection process
- Provide tools/techniques/methodology for quality assurance	Expanded state-of-the-art tools, techniques, and methodologies
- Provide subcontractor control	Can provide measurements on which to base acceptance of deliveries

to assure conformance with user/customer requirements. The software quality metric concepts provide a methodology for the user/customer to specify life-cycle-oriented quality requirements, usually not considered, and a mechanism for measuring if those requirements have been attained.

A function usually performed by quality assurance personnel is a review/audit of software products produced during a software development. The software metrics add formality and quantification to these document and code reviews (Ref 28).

More importantly, the framework provides a means of quantitatively assessing how well the development is processing relative to the quality goals established. This augments current techniques used by quality assurance personnel which may range from only testing to testing and standards enforcement, participation in walkthroughs, etc.

Testing is usually oriented toward correctness, reliability, and performance. The metrics assist in the evaluation of other quality factors like maintainability, portability, and flexibility (Ref 23). The periodic application of the metrics during a large-scale software development effort can be viewed as a control system. Snapshot assessments are generated and feedback to the program management is provided with respect to their specified requirements for quality, thereby allowing corrective action, calibration, redirection, or identification of areas to be emphasized later in the

development, i.e., such as during testing.

The metrics may be in the form of a checklist used to "grade" a document produced during the development of specific attributes such as the number of paths through a module or the number of unconditional branches (Ref 23). In short, Software Quality Metrics provide the quantification needed in a SQA program.

Comparison of Metrics, Inspections, and Walk-Throughs

Software metrics offer more quantification of software quality than other SQA tools and techniques. Software metrics, code inspections, and structured walk-throughs have all been developed to aid the SQA function. Metrics are used throughout the SDLC and quantitatively measure the quality of the software as it is developed (Ref 52). Code inspection techniques are primarily for finding errors in design and code (Ref 29). The primary purpose of structured walk-throughs is to subject the design or code to a critical evaluation (Ref 30).

Currently AFLC uses code inspections and structured walk-throughs. Although the level of usage for these techniques varies for each project, these techniques fail to quantify quality even when both are used to their fullest potential. Table II exposes the gaps in current QA techniques by comparing the properties of Code Inspection, Structured Walk-Throughs, to Software Quality Metrics (Ref 23).

Although Code Inspection compares favorably to

TABLE II

Comparison of Key Properties of Inspections and Walk-Throughs and Metrics

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

Structured Walk-Throughs, it fails to provide quantification of software quality. The quantification of quality is the most favorable attribute of Software Quality Metrics in addition to it matching the other properties of Code Inspection and Structured Walk-Throughs.

Software metrics have both anomaly-detecting and predictive characteristics, in addition to those which may be classified as acceptance metrics (Ref 52). Code inspections and walk-throughs are oriented towards anomaly detection. They are techniques used by development teams; yet metrics not only can be used by the development team, but also can be used by the project manager as acceptance criteria (Ref 23).

Identifying Software Quality Requirements

The vehicle for establishing software quality requirements is the hierarchical model of software quality (see Figure 5, Chapter I) defined in "A Framework for the Measurement of Software Quality", Proceedings of the ACM Software Quality Assurance Workshop in November 1978 (Ref 18).

The procedures for establishing the quality requirements for a particular software system utilize this model and will be described as a three level approach, which are the levels corresponding to the hierarchical levels of the software quality model. The basic tool to be used in identifying the important quality factors will be the Software Quality Requirements Survey form shown in

Table III. The formal definitions of each of the eleven quality factors are provided on that form (Ref 17).

It is recommended that a briefing be provided to the decision makers using the tables and figures, which will be described later, to solicit their responses to the survey. In order to determine the quality factors, the decision makers should consider the system characteristics as shown in Table IV (Ref 23). In other words, if the software system is to have a long life cycle, the Quality Factors that must be considered are Maintainability, Flexibility, and Portability.

Table V shows that the eleven quality factors, which are identified on the survey, can be grouped according to three life cycle activities associated with a delivered software product. These three activities are product operation, product revision, and product transition (Ref 23). The cost to implement versus life cycle cost reduction relationship exists for each quality factor. The benefit versus cost-to-provide ratio for each factor is rated as high, medium, or low in the right hand column of Table V. This relationship and the life cycle implications of the quality factors should be considered when selecting the important factors for a specific system.

By this point, a tentative list of quality factors should be produced. The next step is to consider the interrelationships among the factors selected. Table VI and Table VII can be used as a guide for determining the

Table III. Software Quality Requirements Survey Form

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

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

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

3. Are you currently in:

- _____ 1. Development phase
 _____ 2. Operations/Maintenance phase

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

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

Table IV
System Characteristics and Related Quality Factors

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

Table V
The Impact of Not Specifying or Measuring Software Quality Factors

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

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

Table VI
Relationships Between Software Quality Factors

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

LEGEND

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

○ = High ● = Low

Blank = No relationship or application dependent

Table VII

SOME TRADEOFFS BETWEEN
SOFTWARE QUALITY CHARACTERISTICS

INTEGRITY VS EFFICIENCY--The additional code and processing required to control the access of the software or data usually lengthen run time and require additional storage.

USABILITY VS EFFICIENCY--The additional code and processing required to ease an operator's task or provide more usable output usually lengthen run time and require additional storage.

MAINTAINABILITY VS EFFICIENCY--Optimized code, incorporating intricate coding techniques and direct code, always provides problems to the maintainer. Using modular, instrumented, and well-commented high level code to increase the maintainability of a system usually increases overhead, resulting in less efficient operation.

TESTABILITY VS EFFICIENCY--The above discussion applies to testing.

PORTABILITY VS EFFICIENCY--The use of direct code or optimized software or utilities decreases the portability of the system.

FLEXIBILITY VS EFFICIENCY--The generality required for a flexible system increases overhead and decreases the efficiency of the system.

REUSABILITY VS EFFICIENCY--The above discussion applies to reusability.

INTEROPERABILITY VS EFFICIENCY--Again the added overhead for conversion from standard protocol and standard data representations, and the use of interface modules decreases the operating efficiency of the system.

FLEXIBILITY VS INTEGRITY--Flexibility requires very general and flexible data structures. This increases the data security problem.

REUSABILITY VS INTEGRITY--As in the above discussion, the generality required by reusable software provides severe protection problems.

INTEROPERABILITY VS INTEGRITY--Coupled systems allow for more avenues of access and different users who can access the system. The potential for accidental access of sensitive data is increased as well as the opportunities for deliberate access. Often, coupled systems share data or software which compounds the security problems as well.

relationships between the quality factors (Ref 28). Some factors are synergistic while others conflict. The impact on conflicting factors is that the cost to implement will increase. This will lower the benefit-to-cost ratio. For example, there is a very low relationship between maintainability and efficiency. It would be unrealistic or very costly for a user to expect a software system to be highly maintainable and highly efficient. An efficient system uses optimized code and incorporates intricate coding techniques; as such, it will always provide problems to the maintainer. On the other hand, a highly maintainable system uses modular and well-commented high level code which increases the system overhead, resulting in less efficient operation.

Now a list of quality factors that are considered to be important for one particular system should be compiled. The list should be organized in order of importance. The definitions of the factors chosen should be included with this last list. The rationale for the selection of factors must be documented.

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

Table VIII
Software Criteria and Related Quality Factors

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

software attributes associated with the chosen critical quality factors (Ref 17). For example, if the desired management-oriented quality factors of the system are Reliability and Maintainability, then the software must exhibit the following criteria: Consistency and Simplicity (both common to Reliability and Maintainability) as well as Accuracy and Error Tolerance (attributes of Reliability), and Conciseness, Modularity and Self-Descriptiveness (attributes of Maintainability).

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

Currently, the underlying mathematical relationships that allow measurement at this level of precision do not exist for all the quality factors. The relationships exist for reliability, maintainability, portability, and flexibility (Ref 28). The mechanism for making the precise statement for any quality factor is a rating of the factor. The underlying basis for the ratings is the effort or costs required to perform a function such as to correct or modify the design or program (Ref 23). For example, rating for maintainability might be that the average time to fix a problem should be five man-days or that 90% of the problem fixes should take less than six man-days. This rating would be specified as a quality requirement. To comply with this specification, the software would have to exhibit characteristics which, when present, give an indication

that the software will perform to this rating, i.e., be fixed in less than six man-days if a problem should occur. These characteristics are measured by metrics which are inserted into a mathematical relationship to obtain the predicted rating.

In order to choose ratings such as the two mentioned above, data must be available which allows the decision maker to know what is a "good rating" or perhaps what is the industry average. Currently there is generally a lack of good historical data to establish these expected levels of operations and maintenance performance for software.

There are significant efforts underway to compile historical data and derive the associated performance statistics (Ref 9). Individual software development organizations should attempt to compile historical data for their particular environment.

The Automated Measurement Tool (AMT), which is discussed in Chapter IV, has been created to collect this data. It is designed to be both language and machine independent (Ref 18).

The vehicle for applying the software quality metrics are the metric worksheets contained at the end of this report in Appendix A. Figure 6 illustrates the timing of when each worksheet is to be applied during the SDLC (Ref 23). The procedure is to take the available documentation/source code, apply the appropriate worksheet, and translate the measurements to metric scores. For

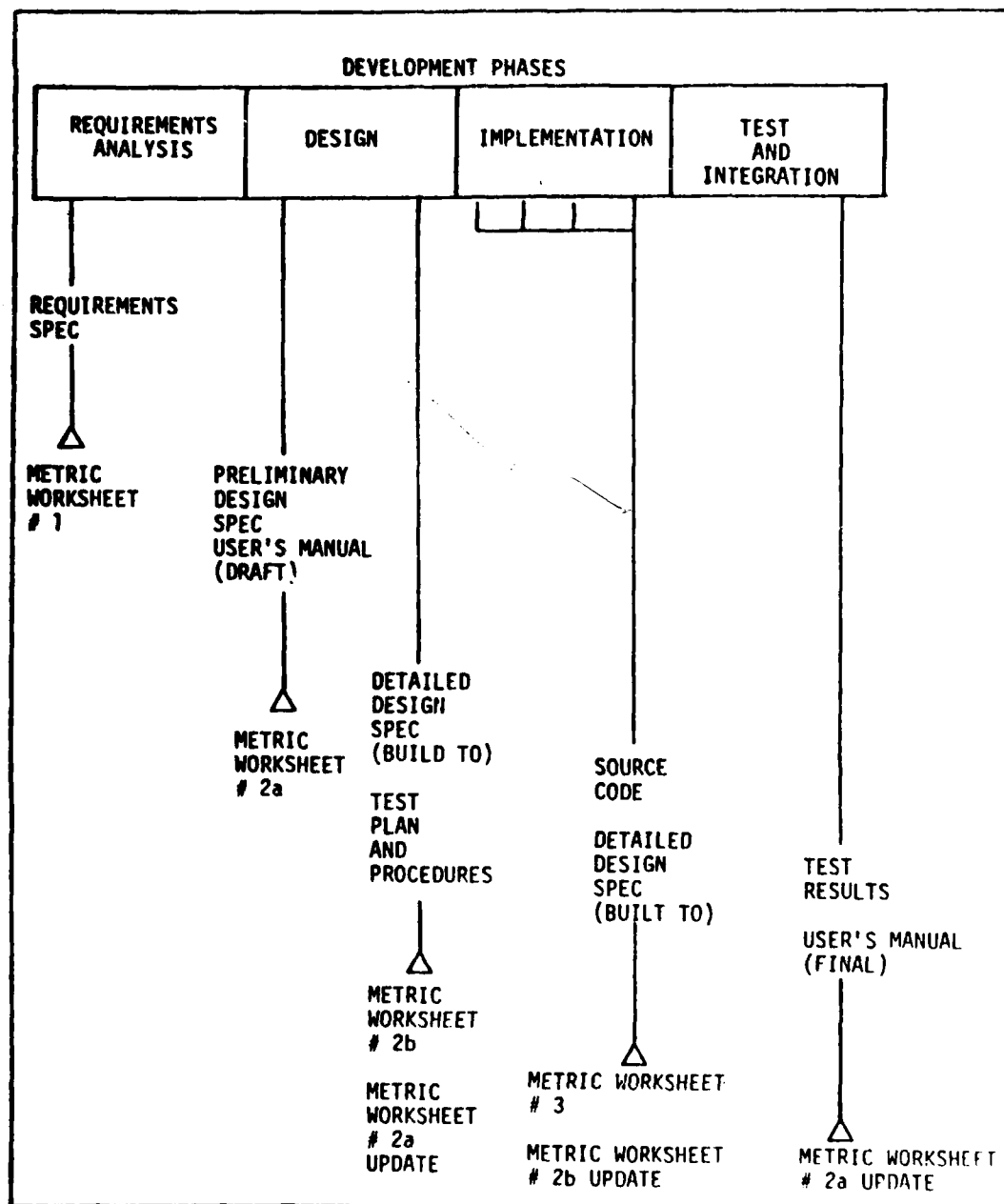


Figure 6.
Application of the Metric Worksheets

example, worksheet #2a would be used during the design phase of a system development effort. This worksheet should be able to collect the needed system level information from the Preliminary Design Specifications and the Draft of the Users Manual. Then the AMT would be updated with this information, and then automatically compute the metric scores.

Techniques for Applying Metrics

The metrics can be applied different ways. The first technique for applying the metrics is by formal inspection.

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

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

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

produce.

These Software Quality Metrics are now sufficiently developed to use within a SQA program. The AMT reduces the requirement for human resources by automating much of the collection of data.

Chapter III presents the methods used to derive the information needed to make recommendations concerning the development of AFLC/LM's Software Quality Assurance Program. It discusses the research required to demonstrate the feasibility of incorporating Software Quality Metrics into AFLC's SQA program.

CHAPTER III

RESEARCH METHODOLOGY

The overall objective of this thesis is to provide guidelines to AFLC/LM concerning the way Software Quality Metrics should be integrated into its Software Quality Assurance program. This chapter will discuss the methodology used to attain this objective and indicate how the four research goals specified in Chapter I were achieved.

Establishing the Framework

In order to accomplish the first goal: To determine which measurement tools and techniques should be used to support a SQA program, a framework for analysis was established by reviewing current Software Quality Assurance programs throughout DoD and the industry. This provided a baseline to determine how software quality has been assessed.

Since it was critical to have a broad source of current information, tutorials and proceedings were read from conferences and symposiums on Quality Management, Quality Control, Quality Assurance, Software Design, Software Management, Software Engineering, and Configuration

Management. Some of the more significant contributions in this survey were: Donald J. Reifer's "Tutorial on Software Management", an IEEE Catalog in 1979 (Ref 7), Freeman and Wasserman's "Tutorial on Software Design Techniques", an IEEE Catalog in 1980 (Ref 8), and Bryan, Chadbourne, and Siegel's "Tutorial on Software Configuration Management", an IEEE Catalog in 1980 (Ref 50).

The tutorials provided between thirty to fifty-five articles each on current topics concerning the software system development life cycle. This review helped provide an overview of what tools and techniques were being effectively used in industry and the government in building Software Quality Assurance programs.

Attendance at the DPMA National Symposium on Effective Methods of EDP Quality Assurance on 1-3 April 81 in Chicago, IL, provided the opportunity to speak to Canadian and U.S. Quality Assurance experts. The full day workshop on "Establishing the Quality Assurance Function" and the two day conference which hosted several QA experts who presented material on current QA efforts and directions provided insights to recent industrial developments in QA.

Membership in ACM and IEEE allowed attendance at their local meetings which host speakers who discuss recent developments in software and hardware. Additionally, membership in four of ACM's Special Interest Groups: SIGSOFT, SIGSIM, SIGCAS, and SIGBDP and two of IEEE's Societies: Engineering Management Society and Computer

Society have provided all their publications which keep members current on developments and practices in the computer field.

Membership in The Library of Computer and Information Sciences Book Club, provided recent publications that the libraries in this area do not have. The books that were purchased cover Quality Control, Project Management, Software Quality Management, Software Engineering, and Structured Requirements Definition.

By September of 1981, most of the exploratory research had been completed. By then it was clear that Software Metrics represented the "leading edge" of SQA because it promised the ability to quantify quality by providing objective measurement (Ref 140).

Model of Development Decision Process

The second goal: To develop a model of the decision making process of AFLC/LM's software system development effort which incorporates the application of Software Quality Metrics, was accomplished in three phases.

Phase 1. During this phase, interviews and additional literature surveys were conducted to determine how current research in systems modeling and software metrics might help in solving AFLC's problems in software development.

Interviews with AFLC/LM personnel were conducted to clearly define the current problems in software development, and to discuss how software metrics might provide some of the solutions by providing a quality

assessment capability. Richard Woodward, the AFLC/LM sponsor of this thesis, provided the initial QA background at AFLC. Cloyd D. Stratton provided information about Configuration Management and its evolution at AFLC/LM. Jack Tinsley provided information which described how CM is functionally carried out, i.e., "who does what" during the SDLC.

Additional literature surveys were conducted to determine who had been doing the research in software metrics and where the software metrics were being applied. This required a review of DoD material on SQA.

Because Rome Air Development Center (RADC) was sponsoring research in the area of software metrics, a point of contact was established with Joe Cavano at Griffis AFB, NY. Results of RADC's testing were not available until March 82, at which time they were requested.

After discovering the efforts of James McCall and others at General Electric's Command and Information Systems Division at Sunnyvale, CA, materials were obtained on their initial findings about Software Quality Metrics. In order to clarify certain issues proposed by this documentation concerning Software Quality Metrics, two days were spent interviewing James McCall and Dave Markham at G.E. on October 29 and 30 to determine the usefulness of SQM and the validity of the metrics that they had derived. Since G.E.'s effort to derive and validate the Software Quality Metrics is of obvious importance to the usefulness

of SQM as a measurement tool in a SQA program, a detailed discussion of G.E.'s efforts has been included in this thesis.

Phase 2. This phase involved developing a model which displays the interaction between the software development decision-making process and the user environment using QA and systems concepts acquired in six AFIT courses: EE 5.45--Software Systems Acquisition, EE 6.99B--Software Management Information Requirements, EE 6.93--Software Engineering, SM 6.44--Techniques in Project Management, LM 6.15--Logistics Decision Support Systems, and MA 5.48--Computer Systems Analysis.

The model represents the SDLC decisioning process. It illustrates the interaction between the software development management organization (Project Management including the SQA function) and the user environment. Moreover, each subsystem within this model (scanning, intelligence, and decision) is displayed in a manner that clearly demonstrates the interrelationships between its input, output, process, and feedback mechanisms.

Phase 3. Finally, in order to superimpose the Software Quality Metrics framework onto the Phase 2 conceptual model, the procedures developed by General Electric for applying the metrics were "fit" into the decision-making process to reveal how the metrics, by providing quantified measures of software quality during each phase of the SDLC, can improve the end-product of the software development.

Metric Application

The third goal: To demonstrate how the SQM concepts can be applied on an existing system at AFLC, was achieved by the development of checklists which are to be used by system developers. Documentation was acquired through Robert Brooks, AFLC/LMX, who is the Project Coordinator for the Maintenance Management Systems Improvement Project (MMSIP). Through Mr. Brooks, the documentation on G072A MMSIP, which is a subsystem of MMSIP, was obtained (Ref 142).

The procedures and metrics worksheets, which were developed by General Electric, were applied to this subsystem. Because the subsystem had just become operational, the development documentation, which is described in Appendix F, was still available.

To satisfy the request of the sponsor, AFLC/LM, the real intent of this third goal was to indicate what changes, if any, had to be made to current documentation requirements in order to utilize the Software Quality Metrics in a Software Quality Assurance program for AFLC/LM. Therefore, checklists, which account for the software-oriented requirements for Reliability and Maintainability, were developed to insure that software developers know in advance that specific items should be considered in developing reliable and maintainable software.

By fulfilling this request, AFLC/LM will be able to acquire the Automated Measurement Tool (AMT) from Rome Air Development Center and already be set-up to use the tool in their SQA programs.

Metric Integration

The achievement of the fourth goal: To use the information generated from this research to help AFLC/LM formulate guidelines to integrate Software Quality Metrics and other software tools into a Software Quality Assurance program, allowed recommendations to be made concerning:

1. what changes in current Configuration Management practices, such as in documentation of the system development life cycle, should be made to meet the information requirements of measurement tools;
2. what software tools should be acquired to support the collection of SQM;
3. to what extent the Software Quality Metric framework could be used if the automated tools are not acquired; and finally
4. what further applications can be seen for the use of metrics.

Chapter IV accomplishes the first goal of this thesis by reviewing current research in Software Quality Assurance (SQA) and analyzing software measurement tools and techniques.

CHAPTER IV

SQA TOOLS AND TECHNIQUES: A SURVEY OF DOD AND INDUSTRY

A Review of current Software Quality Assurance (SQA) programs throughout DoD and the industry has provided a baseline to determine how software quality has been assessed. The initial literature search revealed a diversified description of software attributes. Table IX summarizes the list of software attributes and indicates the literature source for each attribute. Appendix E provides the various definitions for each software attribute, as it is described in the literature.

This initial survey demonstrated the need for a diversified set of SQA tools and techniques that would be required to assess the characteristics of a software system. Related research by General Electric has grouped these software attributes into eleven software factors, with the related attributes serving as criteria which further define each factor. This consolidation of software characteristics is supported by other research which has used these factors to make comparisons of SQA tools and techniques (Refs 15, 23).

The goal of this chapter is to determine which measurement tools and techniques should be used to support

Software Attributes Identified in the Literature

SOFTWARE ATTRIBUTES	AUTHORS	
	Boehm [19]	Casey [59]
ACCEPTABILITY	.	.
ACCESSIBILITY	.	.
ACCOUNTABILITY	.	.
ACCURACY	.	.
ADAPTABILITY	.	.
AUGMENTABILITY	.	.
AVAILABILITY	.	.
CLARITY	.	.
COMMUNICATIVENESS	.	.
COMPATIBILITY	.	.
COMPLETENESS	.	.
COMPLEXITY	.	.
CONCISENESS	.	.
CONSISTENCY	.	.
CONVERTIBILITY	.	.
CORRECTNESS	.	.
COS	.	.
DOCUMENTATION	.	.
EFFICIENCY	.	.
EXPANDABILITY	.	.
EXPRESSION	.	.
EXTENSIBILITY	.	.
FLEXIBILITY	.	.
GENERALITY	.	.
HUMAN FACTORS	.	.
INTEGRITY	.	.
INTEROPERABILITY	.	.
LEGIBILITY	.	.
MAINTAINABILITY	.	.
MANAGEABILITY	.	.
MODIFIABILITY	.	.
MODULARITY	.	.
OPERABILITY	.	.
PERFORMANCE	.	.
PORTABILITY	.	.
PRECISION	.	.
PRIVACY	.	.
RELIABILITY	.	.
REPAIRABILITY	.	.
REUSABILITY	.	.
ROBUSTNESS	.	.
SECURITY	.	.
SELF-CONTAINEDNESS	.	.
SELF-DESCRIPTIVENESS	.	.
SERVICEABILITY	.	.
STABILITY	.	.
STRUCTUREDNESS	.	.
TESTABILITY	.	.
TIME	.	.
TOLERANCE	.	.
TRANSFERABILITY	.	.
UNDERSTANDABILITY	.	.
UTILITY	.	.
VARIABILITY	.	.

a SQA program for AFLC/LM. To accomplish this goal, categories for software QA tools and techniques (aids) have been presented. Next, material on "toolsmithing" has been provided, and an assessment of state-of-the-art technology in SQA aids has been made. Finally, a discussion about the use of Software Quality Metrics (SQM) and the Automated Measurement Tool (AMT) concludes the chapter.

Techniques, Tools, and Methodologies

Experience has indicated that good techniques, tools, and methods must be employed to have a successful quality assurance program. Software QA is a service function created to provide management with the independent checks and balances it needs to control and assess the software system quality. Economic considerations are such that the SQA function must accomplish its assigned task efficiently and at minimum cost to remain effective. Technical considerations are such that SQA can only justify its role if it contributes directly to the system development with minimal disruption and meaningful results. Both economic and technical considerations dictate that SQA use proven tools to automate the techniques and methods it employs to accomplish its purpose (Ref 15).

For the purpose of this thesis, automation is defined as mechanizing methods using tools and techniques. Tools are computer programs used to aid the quality inspector in evaluation of the developer's software system. Techniques

are procedures arranged to simplify the evaluation process (Ref 15).

As shown in Table X, SQA Tools and Techniques, software QA personnel employ the methods of inspection, analysis, demonstration, and test. Inspection confirms product or procedure compliance with stated requirements by examination. Analysis studies the product or procedure in detail in order to analytically confirm an answer or results of a solution. Demonstration provides tangible and visible evidence of compliance by making trial output acceptable for review and comparison against stated goals.

Software QA tools and techniques can be classified based upon the method they support. Table X lists available tools and techniques by category or class. Appendix H provides a glossary for these SQA tools and techniques. It should be noted that only the Software Quality Metrics techniques and the Automated Measurement Tool (AMT) support the full spectrum of SQA methods.

Techniques. Table XI illustrates which techniques support the SQA functions. These functions are representative of QA functions that are specified in MIL-S-52779A, Software Quality Assurance Program Requirements (Ref 86). As shown in the table, Software Quality Metrics, Reviewing, Auditing, and Standardization all support the full range of SQA functions. AFLC/LM incorporates reviews and audits throughout the software development life cycle (SDLC) under Configuration

Table X
SQA Tools and Techniques

Class	Inspection	Analysis	Demonstration	Test
Tech- nique	Auditing Code inspection Design inspection Reviewing Software Quality Metrics (SQM)	Analytical modeling Correctness proof Error-prone analysis Execution analysis Post-functional analysis Simulation Software Quality Metrics (SQM) Standardization Static analysis	Functional testing Software Quality Metrics (SQM) Walk-throughs	Algorithm evaluation test Correctness proofs Equivalence classes Functional testing Logical testing Path testing Simulation Software Quality Metrics (SQM) Stress testing Symbolic execution
Tool	Automated Measurement Tool (AMT) Consistency checker Editor Requirements tracer Standards analyzer	Accuracy study processor Automated Measure- ment Tool (AMT) Comparator Consistency checker Cross-referencers Data base analyzer Decision tables Dynamic analyzer Editor Flowchart Hardware monitor Interface checker Interrupt analyzer Logic analyzer Simulators Software monitor Static analyzer Structure analyzer Timing analyzer	Automated Measure- ment Tool (AMT) Dynamic simulator Hardware monitor Software monitor Standards analyzer Test bed	Automated Measure- ment Tool (AMT) Automated test generator Comparator Debugger Dynamic analyzer Dynamic simulator Flowchart Hardware monitor Instruction trace Simulators Software monitor Test drivers, scripts Test-result processor

Table XI
Supporting Techniques for SQA Functions

Function										
Techniques	Quality planning	Work tasking	Configuration management	Testing	Corrective action	Library controls	Design standards	Documentation standards	Reviews and audits	Subcontractor controls
1. Algorithm evaluation test				X						
2. Analytical modeling				X						
3. Auditing				X						
4. Code inspection				X						
5. Correctness proof				X						
6. Design inspection				X						
7. Error-prone analysis				X						
8. Equivalence classes				X						
9. Execution analysis				X						
10. Functional testing				X						
11. Logical testing				X						
12. Path testing				X						
13. Post-functional analysis				X						
14. Reviewing				X						
15. Simulations				X						
16. Standardization				X						
17. Software Quality Metrics				X						
18. Static analysis				X						
19. Stress testing				X						
20. Symbolic execution				X						
21. Walk-throughs				X						

Management (CM), and it is striving to improve its procedures for standardization (Ref 38). Software Quality Metrics (SQM) provides the procedures which incorporate Reviewing, Auditing, and Standardization, and it establishes guidelines, in the form of checklists, which enhance other SQA techniques in use (Ref 23).

Table XII reveals the effectiveness of SQA techniques in assessing quality properties. The techniques that possess the highest assessment capability for all the quality factors are: Design Inspection, Software Quality Metrics, and Standardization (Walk-throughs provide "ease in use" as well as effectiveness in assessment).

Tools. Table XIII depicts the tools that support various SQA functions that are specified in MIL-S-52779A (Ref 86). The Automated Measurement Tool (AMT), Standards and Text Editors are the only tools which support the full spectrum of SQA functions (Refs 15, 31).

Table XIV displays the effectiveness each tool provides for assessing software quality. Only the AMT, Standards, and Test Bed tools are highly effective in assessing all quality properties. The Language Processor, Standards Analyzer, and Dynamic Analyzer also provide effectiveness to SQA program (Refs 15, 84).

Toolsmithing

As can be ascertained from the previous section, the software quality engineer has many tools and techniques at his disposal. One of the quality engineer's major tasks in

Table XII
Technique Effectiveness in Assessing Quality Properties

Techniques	Quality property								
	Correctness	Efficiency	Integrity	Maintainability	Modifiability	Portability	Reliability	Testability	Usability
1. Algorithm evaluation test	M	M	M	M	M	M	H	M	M
2. Analytical modeling	M	M	M	M	M	M	H	M	M
3. Auditing	M	M	M	M	M	M	H	M	M
4. Code inspection	M	M	M	M	M	M	H	M	M
5. Correctness proof	M	M	M	M	M	M	H	M	M
6. Design inspection	M	M	M	M	M	M	H	M	M
7. Error-prone analysis	M	M	M	M	M	M	H	M	M
8. Equivalence classes	M	M	M	M	M	M	H	M	M
9. Execution analysis	M	M	M	M	M	M	H	M	M
10. Functional testing	M	M	M	M	M	M	H	M	M
11. Logical testing	M	M	M	M	M	M	H	M	M
12. Path testing	M	M	M	M	M	M	H	M	M
13. Post-functional analysis	M	M	M	M	M	M	H	M	M
14. Reviewing	M	M	M	M	M	M	H	M	M
15. Simulation	M	M	M	M	M	M	H	M	M
16. Software Quality Metrics	M	M	M	M	M	M	H	M	M
17. Standardization	M	M	M	M	M	M	H	M	M
18. Static analysis	M	M	M	M	M	M	H	M	M
19. Stress testing	M	M	M	M	M	M	H	M	M
20. Symbolic execution	M	M	M	M	M	M	H	M	M
21. Walk-throughs	M	M	M	M	M	M	H	M	M

Legend: H - High effectiveness M - Medium effectiveness L - Low effectiveness

Table XIII

Supporting Tools for SQA Functions

Tools	Function									
	Quality planning	Work tasking	Configuration management	Testing	Corrective action	Library controls	Design standards	Documentation standards	Reviews and audits	Subcontractor controls
1. Accuracy study processor				X						
2. Automated Measurement Tool	X	X	X	X	X	X	X	X	X	X
3. Automated test generator				X						
4. Comparator			X	X				X		
5. Consistency checker							X	X		
6. Cross-reference				X						
7. Data base analyzer				X						
8. Debugger				X						
9. Decision tables		X	X	X	X	X			X	X
10. Dynamic analyzer				X						
11. Dynamic simulator				X						
12. Editor				X						
13. Flowcharter				X			X			
14. Hardware monitor				X						
15. Instruction trace				X						
16. Interface checker				X						
17. Interrupt analyzer				X						
18. Language processor				X			X			
19. Libraries			X	X						X
20. Logic analyzer				X						
21. MIS		X	X	X	X	X				X
22. Requirements tracer			X	X	X				X	X
23. Simulator				X			X			
24. Software monitor				X						
25. Standards	X	X	X	X	X	X	X	X	X	X
26. Standards analyzer				X			X	X		
27. Static analyzer				X			X	X		
28. Structure analyzer				X			X			
29. Test bed				X						
30. Test drivers, scripts				X						
31. Test-result processor				X						
32. Text editor	X	X	X	X	X	X	X	X	X	X
33. Timing analyzer				X						

Table XIV

Tool Effectiveness in Assessing Quality Properties

Quality Property									
Tools	Correctness	Efficiency	Integrity	Maintainability	Modifiability	Portability	Reliability	Testability	Usability
1. Accuracy study processor	M	L	L	L	L	L	H	L	L
2. Automated Measurement Tool	H	H	H	H	H	H	H	H	H
3. Automated test generator	M	L	L	L	L	L	H	H	L
4. Comparator	L	L	L	L	L	M	M	M	L
5. Consistency checker	H	L	L	L	L	L	M	L	L
6. Cross-reference	M	L	L	M	M	L	M	L	L
7. Data base analyzer	M	L	L	M	M	M	M	M	L
8. Debugger	M	M	L	L	L	L	H	L	L
9. Decision tables	M	L	M	L	L	L	M	M	L
10. Dynamic analyzer	H	H	L	M	M	L	H	M	L
11. Dynamic simulator	H	M	L	L	L	L	H	L	L
12. Editor	M	L	L	M	M	L	M	M	L
13. Flowcharter	H	L	L	M	M	M	M	H	H
14. Hardware monitor	M	H	L	L	L	L	H	M	L
15. Instruction trace	L	M	L	L	L	L	H	M	L
16. Interface checker	H	L	M	M	M	L	H	L	L
17. Interrupt analyzer	H	L	M	M	M	L	H	L	L
18. Language processor	H	M	M	H	H	H	H	H	H
19. Libraries	L	L	H	L	L	L	L	L	H
20. Logic analyzer	M	M	L	L	L	L	H	M	L
21. MIS	H	L	L	L	L	L	L	L	L
22. Requirements tracer	H	L	L	M	M	M	M	H	L
23. Simulator	H	H	M	M	M	M	M	M	H
24. Software monitor	M	H	L	L	L	L	H	M	L
25. Standards	H	H	H	H	H	H	H	H	H
26. Standards analyzer	H	L	M	H	H	M	M	M	M
27. Static analyzer	H	L	M	M	M	L	M	M	M
28. Structure analyzer	H	M	L	M	M	M	M	M	M
29. Test bed	H	H	H	H	H	H	H	H	H
30. Test drivers, scripts	M	L	L	L	L	L	H	H	L
31. Test-result processor	M	L	L	M	M	M	M	H	H
32. Text editor	M	L	L	H	H	L	L	H	H
33. Timing analyzer	H	L	M	M	M	L	H	L	L

Legend: H - High effectiveness M - Medium effectiveness
L - Low effectiveness

developing an acceptable software quality assurance program for a specific organization and/or project is to select those tools and techniques that will allow him to accomplish his functions in the most efficient and cost-effective manner. Selection assumes that the quality assurance organization has its methods and techniques codified in written form (i.e., usually as procedures in a manual), its tools developed and maintained under configuration control in a centralized library, and its sources for supplementing existing tools and techniques identified in case acquisition is warranted (Ref 15). Factors which impact selection and should be evaluated are listed in checklist form in Table XV (Ref 87).

This checklist should be used to evaluate each tool and technique that is considered for inclusion in a SQA program. Using this checklist to assess Software Quality Metrics (SQM) and the Automated Measurement Tool (AMT) as possible aids in the SQA program for AFLC/LM, a manager or software quality engineer will quickly realize that SQM and AMT should be acquired because every response on the checklist is affirmative:

Applicability. The AMT and SQM are suited for the task. They provide the needed measures, and already check the quality of COBOL source code (Ref 84).

Availability. The SQM and AMT are now ready for use. Rome Air Development Center completed testing and validation in March 1982. The AMT was prototyped on the VAX 11/780 and modified to be used by the RADC H6180 (Ref 84).

Cost/Benefit. The software and tapes for the AMT and the supporting documents for SQM can now be acquired by DoD agencies by submitting the request to RADC.

Table XV
Factors Impacting Tool and Technique Selection

Factor	Evaluation criteria	Yes	No	Notes
Applicability	Is the tool or technique suited for the task? If not, can it be modified easily to do the job?			
Availability	Is the tool or technique ready for use? If so, can it be delivered to the customer in acceptable form if he desires it?			
Cost/Benefit	Can the use of the tool or technique be justified in terms of return on investment? If not, why use it?			
Experience	Has the tool or technique been used on other projects? If so, how many and with what results? If not, have adequate precautions been taken to manage the potential risk?			
Quality	Are the tools and techniques documented and is adequate user documentation available? Are the tools under configuration control? Have the tools been qualified formally? Have the tools been developed using modern programming techniques and a High Order Language (HOL)?			
Resources	Have the potential sizing and timing growth implications of tool use been factored into the decision?			
Risk	Have the risks associated with developing or using the tool or technique been quantified?			

Experience. The SQM and AMT have been used on both Army and Air Force projects. The documentation provides the strengths and current limitations (Ref 23).

Quality. The SQM and AMT are well-documented, are under configuration control, and have been qualified through DoD contracts. The implementation of the AMT was done in IFTRAN, a General Research Corporation structured programming preprocessor to FORTRAN (Ref 89).

Resources. The SQM and AMT documentation provide the information needed for conversion to the AFLC environment. However, further expansion of metric application could be achieved with development assistance by G.E.

Risk. The only risk involved in using SQM and the AMT is if AFLC does not first develop a database of operational history of its systems before relying on the predictive accuracy of the metric scores. The AMT can be used to build this data base (Ref 51).

Based on the previous discussion of SQA aids, it is believed that the tools and techniques displayed in Table XVI represent the minimum set required to construct a responsive and quantitative SQA program (Refs 15, 24). This set provides for a balanced coverage of the required SQA functions as illustrated in Tables XI and XIII. It also provides for a balanced assessment of the software quality factors as illustrated in Tables XII and XIV.

Table XVI
Minimum Set of Tools and Techniques

Techniques		Tools	
Software Quality Metrics (SQM)		(AMT)	Automated Measurement Tool
Reviewing			Standards
Auditing			Standards Analyzer
Design Inspections			Dynamic Analyzer
Walk-Throughs			Language Processor
Standardization			Test Bed

Other tools and techniques may be added as the need arises, so long as their cost can be justified in terms of benefits derived. The quantitative aspect of this minimum set of tools and techniques is a direct result of the inclusion of SQM and the AMT.

Chapter II provides the overview for using SQM; Chapter V presents a detailed application of the SQM concepts throughout the SDLC, and Appendix J discusses the procedures for quantitatively assessing software quality by using the lowest level of SQM--the metrics. However, from a management perspective of the SDLC, it is the automated measurement services, which operationalize the metric concepts, that should prompt AFLC/LM to acquire the AMT.

Automated Measurement Services

The concept behind SQM is to provide software managers with a mechanism to quantitatively specify and measure the level of quality in a software system. To provide this mechanism, a software developer must collect data from the products of the software development process. This raw data is then used to calculate metric values which can be used to assess the quality of the software as it is being produced. The purpose of the Automated Measurement Tool (AMT) is to provide automated support to the application of the SQM concepts. Without the AMT, the metrics data must be collected by hand in a tedious, error-prone, and time-consuming process (Ref 84).

The amount of data that can be automatically collected

by the AMT is limited to data that can be derived from machine readable sources such as design materials generated on the computer and the actual source code. The current version of the AMT automatically collects and stores raw data from COBOL source code (most of AFLC/LM's software is written in COBOL). The remainder of the raw data required to calculate the metrics must be manually collected.

A SQA analyst uses the worksheets in Appendix A to answer questions about the system being measured. When the worksheet is complete, its data is entered into the AMT database, where it is stored along with the automatically collected data.

Currently the AMT collects data from COBOL source code for individual modules. A total of 25 different measurements are collected automatically. These measurements can be divided into the following broad classes:

1. Counts of total number of code statements and comment statements;
2. Counts for individual types of statements, i.e., input, output, exit, entry, declarative, etc.;
3. Counts of different types of branching statements both conditional and unconditional, and
4. Counts of operands and operators, for use in calculating Halstead's measure (Ref 23).

The specific data items automatically collected are shown in Table XVII (Ref 84). Also noted in the table is the entry on the worksheet that the item relates to and the metric that it helps calculate. The worksheet entry is identified by worksheet number (WS#), section number (S#), and item number (I#). Thus a notation such as WS3, S1, I1

Table XVII. Automated Metric Data

Data Item	Worksheet Entry	Metric
1. Number of lines of code	WS3,SI,11	MU.2 Modular Implementation
2. Number of lines of code minus comment statements	WS3,SI,12	SI.4 Coding Simplicity
3. Number of declarative statements	WS3,SI,14	SI.4 Coding Simplicity
4. Number of data manipulation statements	WS3,SI,15	SI.4 Coding Simplicity
5. Number of statement labels	WS3,SI,16	SI.4 Coding Simplicity
6. Number of entrances to modules	WS3,SI,17	SI.1 Design Structure
7. Number of exits from modules	WS3,SI,18	SI.1 Design Structure
8. Maximum nesting level	WS3,SI,19	SI.4 Coding Simplicity
9. Number of decision points	WS3,SI,110	SI.3 Complexity
10. Number of subdecision points	WS3,SI,111	SI.3 Complexity
11. Number of conditional branches	WS3,SI,112	SI.4 Coding Simplicity
12. Number of unconditional branches	WS3,SI,113	SI.4 Coding Simplicity
13. Number of loops	WS3,SI,114	SI.4 Coding Simplicity
14. Number of module modifying statements	WS3,SI,117	SI.4 Coding Simplicity
15. Number of operators	WS3,SI,11	CO.1 Conciseness
16. Number of operands	WS3,SI,12	CO.1 Conciseness
17. Number of unique operators	WS3,SI,13	CO.1 Conciseness
18. Number of unique operands	WS3,SI,14	CO.1 Conciseness
19. Number of comments	WS3,SI,11,11	SD.1 Quantity of Comments
20. Number of continuation lines	WS3,SI,11,113	SD.3 Language Descriptiveness
21. Number of input statements	WS3,SV,11	MI.1 Machine Independence
22. Number of output statements	WS3,SV,12	MI.1 Machine Independence
23. Number of calls to modules	WS3,SV,11	MO.2 Modular Implementation
24. Number of mixed mode expressions	WS3,SV,11,11	EE.3 Executive Efficiency
25. Number of variables initialized when declared	WS3,SV,11,12	EE.3 Executive Efficiency

is read as worksheet 3, section I, item 1.

This automated support counts for 25 of the 92 worksheet #3 data items, or 27% of the measurements required at the implementation phase of a software development. These 25 data items help calculate 9 of the 38 metrics related to implementation, or 24% of those metrics. The metric calculation is described later under Report Generation Services (Ref 17).

The automated data collection is performed by the Automated Measurement Services (AMS) and the Preprocessing Service (PPS) subsystems. The user invokes these subsystems using the MEASURE command. The Preprocessing Services uses an (1) generalized parser to decompose the COBOL source code contained in the file identified by the MEASURE command. The result of the parsing is a parse tree representation of the source code. The AMS subsystem then traverses the parse tree and counts the various data items and enters them in the data base.

The significant aspect of this approach is that other language grammars can be described to the parser, a scanner developed, and the parser will produce a parse tree representation of the other language. With careful attention paid to the token representation, the AMS will be able to process this parse tree representation of the other language with no modification.

In addition to it being language independent, the AMT was developed with the concept of eventually interfacing it

to other software tools in a software development environment. The interfacing would be done by extracting metric data available from the processing done by the other tools and inserting the data into the AMT database so that metrics could be calculated.

A program would have to be written which extracts the appropriate data from the output file of a tool, and using the AMT PUT command, to then insert the data into the database. Potential tools that should be considered are: Requirements Specification Language processors/analyzers, Program Design Language processors/analyzers, code auditors, code instrumentors, and Configuration Management tools. Appendix I provides a tool survey of specific SQA tools that could be considered (Ref 84).

Report Generation Services

The Report Generation Services (RGS) provides the user the ability to generate various reports that reflect the contents of a database. Nine reports may be requested by the user to display the metric data in a variety of formats, and by performing additional calculations, present various forms of data both at the module and system levels. Basically, data is extracted from the database to calculate metric values. The algorithms for performing these calculations, which are listed in Appendix D, are contained in the RGS routines. Samples of the AMT reports are found in Appendix K, and a brief description of each report follows (Ref 84):

MODULE REPORT. This report displays the catalog of modules that have been entered into the database. It provides a status report on the database.

METRIC REPORT. This report calculates the value of each metric categorized by factor and by development phase. This report is used to determine a total picture of the project as measurements are taken.

EXCEPTION REPORT. The exception report delivers the relationship of each module to a given threshold value of a particular metric. The relationship (less than, equal to, or greater than) and the threshold value is input from the user. This report can be used to identify modules whose scores do not meet a certain threshold, identifying them as potential problems.

QUALITY GROWTH REPORT. When the user wishes to track the value of a particular metric over time, the Quality Growth Report will furnish a tabular display of the scores of a selected metric over the phases of the project. This report is used to track a particular metric through a project to see how its value changes.

NORMALIZATION REPORT. The Normalization Report provides the user with the overall rating of a selected quality factor. A series of regression equations are displayed which have been empirically derived from research. The current metric values are substituted in the equations and a rating for the selected quality factor is calculated. Regression equations exist for the quality

factors Reliability, Maintainability, Portability, and Flexibility only.

STATISTICS REPORT. The Statistics Report provides a profile of COBOL constructs for each module.

SUMMARY REPORT. The summary report provides a summary of the metric scores for all of the modules in the system.

WORKSHEET REPORT. The worksheet report displays the raw data entered in each worksheet. It represents the current values in the database. It is used to verify and track data entry.

MATRIX REPORT. This report displays the average and standard deviations for all metrics values for all modules. This report displays all of this information in a matrix form allowing the user to easily identify modules with metric scores that vary from the system average.

The reports may be classified as to their primary use: Descriptive, Historical, and Diagnostic. The reports that are descriptive are the Summary, Matrix, Module, and Metric reports. Their common characteristic is that they report data and in content or format implying no judgements concerning the data. The Summary Report reports all metric scores for each module or for all the modules. It is a detailed and relatively long report. The Matrix Report displays the mean and standard deviation of the modules for each metric. It is a good snapshot of the data in the data base. The Module Report is meant for operational personnel. It simply reports those names of the modules

which are in the data base. The Metric Report is a more detailed output which displays the metric values for each module in a detailed form.

The Historical Reports are the Quality Growth and Worksheet Reports. The Quality Growth report provides the quality trend of a module through the development phases. The Worksheet Report gives a very detailed display of the raw data before it is transformed into metric scores. Its main use is to track data entry and updates.

The Diagnostic Reports are those that show potential problem areas. They are the Normalization, Exception, and Statistics Reports. The Normalization Report applies the regression equations derived from research to metric values related to the quality factors of flexibility, maintainability, portability, and reliability. These regression equations have been developed through examination of previous projects and correlating their metric data with various time and cost data (Ref 23).

Regression equations for the remaining quality factors have not been established. The Exception Report provides a comparison of the metric scores with predetermined, user supplied values. The Statistics Report gives a diagnostic snapshot of any module. These data may be used to evaluate standards or identify potential problem areas (Ref 84).

Table XVIII indicates the reports that support the decisions and actions related to program management, developers, software quality assurance, and test. Some

Table XVIII. Support To Personnel

PERSONNEL	SUPPORT	REPORT TYPE
Program Management	Progress with Respect to Quality Goals	Normalization Function Quality Growth Metrics
Developers	Standards Enforcement Design Decisions	Metrics Summary
Quality Assurance	Standards Enforcement Compliance with Quality Goals Problem Identification	Metrics Quality Growth Exception Normalization Function
Test	Test Strategy Test Emphasis Test Effort	Metrics Summary Exception

examples are:

The Normalization Report calculates an overall rating of a particular quality factor such as reliability and provides program management a summary of the project's progress toward a specific goal for that factor that was set at the outset of the project.

The Metric Report provides the current metric scores. Developers utilize this report to aid in design and implementation decisions.

The Exception Report is utilized by Quality Assurance personnel to identify those modules which vary significantly in characteristics from the average. These modules should be investigated further for non-compliance with standards and conventions or potential problems.

The Summary Report is utilized by test personnel to evaluate the amount of code being developed, the complexity in terms of number of paths, the number of interfaces, etc., any of which have an influence on test strategy and effort.

The adherence to SQM methodology will greatly enhance the effectiveness of the tool to support the production of quality software (Ref 88).

Chapter Summary

The software quality assurance program for AFLC/LM should utilize tools and techniques whose cost/benefits can be demonstrated. The minimum set of these SQA aids that is recommended is shown in Table XVI. The SQA organization should have a toolsmith whose sole responsibility should be the development and maintenance of written procedures and a tool library. Tools placed within the library should be qualified, documented, and placed under configuration control. Acquisition of new tools should occur only when they can be economically justified.

Reifer justified the inclusion of Software Quality Metrics and the Automated Measurement Tool in a SQA program. He stated, "We can conclude from experience that tools and techniques improve final program quality. Before we can progress further, we have to have a better and more quantitative understanding of just what quality is and how you can measure it. Only then can we hope to make quality an integral part of the software systems we produce." (Ref 15) SQM and the AMT provide for the quantitative measurement of software quality that is required.

Chapter V provides the map for utilizing SQM and the AMT by demonstrating their application in the SDLC.

CHAPTER V

SDLC DECISION-MAKING MODEL: A MAPPING TO SQM

It is an accepted proposition that man's judgement is no better than his information, yet the magnitude of data generated by staffs and computers usually only serves to overwhelm the decision-making process. Hence, there is a need to model the decisioning process within the software system development organization to determine what information is needed at specific points during the SDLC.

The Software Development Life Cycle

Before the model can be developed, it is necessary to further describe the Software Development Life Cycle (SDLC) and to identify the deliverables for each phase. Gustafson and Kerr in "Some Practical Experience with a Software Quality Assurance Program" in the Jan '82 Communications of the ACM noted that one of the first steps in establishing a SQA program is "to define the development life cycle used (or preferred) within the organization sponsoring a new quality assurance program..., (and then) the deliverables for each phase must be identified (Ref 56).

Figure 7 provides the details for Software System Development Life Cycle (SDLC) for AFLC/LM. Abstracting

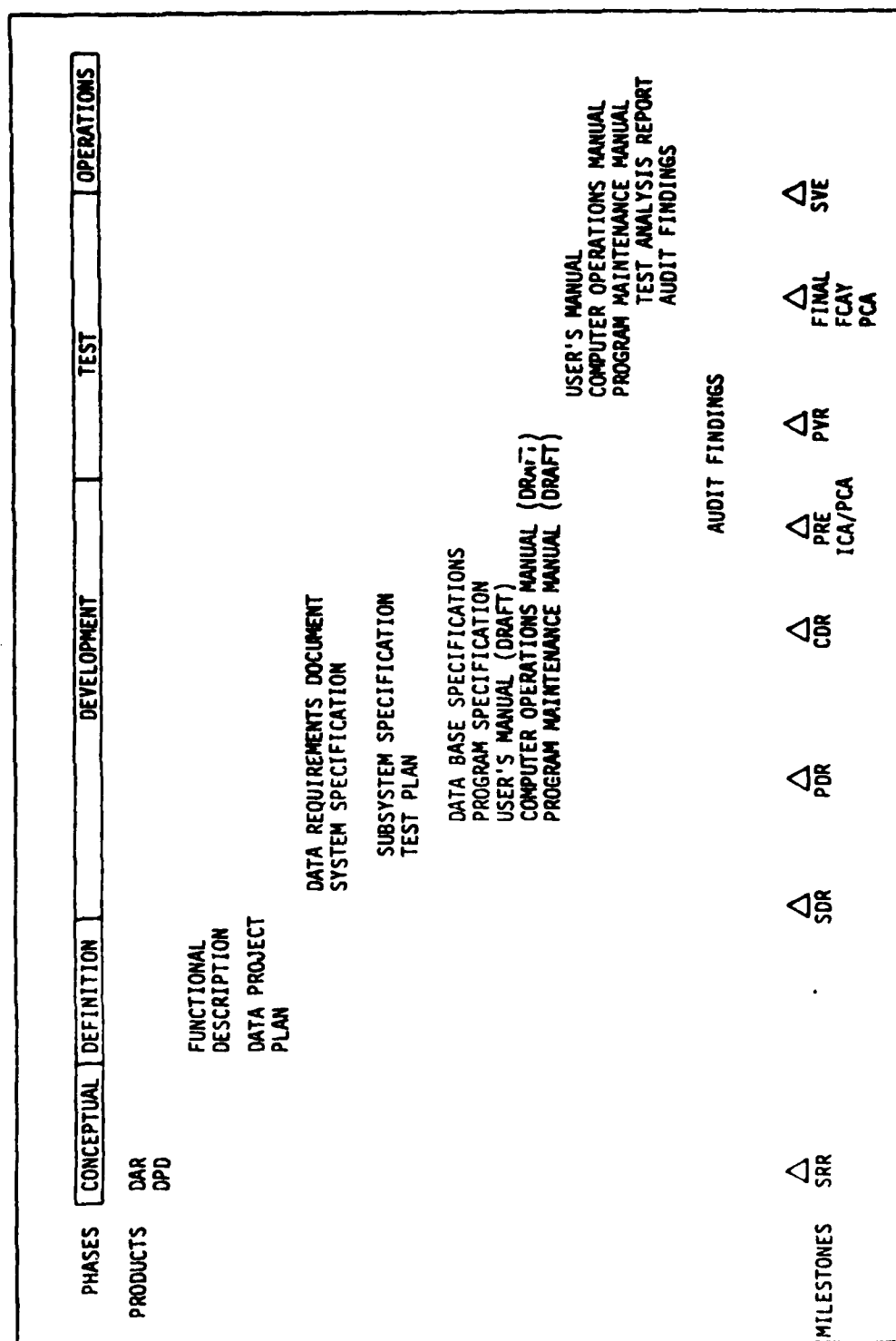


Figure 7. AFIC'S Software System Development Life Cycle

information from the June 1981 revision of AFLC/LM's Configuration Management chart for Automated Data Systems (ADS) Development, Figure 7 defines the phases of the SDLC and identifies the deliverable products for each phase. Moreover, the milestones which indicate the completion of each phase are also identified. For example, the end of the Definition Phase is marked by the System Design Review (SDR), which is a milestone of the SDLC. The primary deliverables are the Functional Description and the Data Project Plan. Appendix F furnishes: the description of each phase of the SDLC, the identification and contents of the deliverable products, and the significance of each milestone.

The deliverable products for the reviews and audits will be used as inputs to the model of the decision-making process within AFLC/LM's software development organization.

Modeling the Decisioning Process

A basic process of the system development organization, which is a basic process of all open systems, is acquiring data from the user environment, and from its internal parts, to be "consumed" in problem-definition and decisioning in the service of its attempts to alter its intended-states-of-affairs, its internal structure or function, for some aspect or domain of its environment (Ref 53).

This process, that of data acquisition, from the environment, constitutes what has here been called the

"scanning process." The system must be related to the environment through two kinds of channels: the afferent (a scanning system), through which it receives information about the environment, and the efferent (decision system), through which it acts on the environment (Ref 54).

The relationships between the software system development organization (project management) and user environment are shown in Figure 8. For logical completeness, a third subsystem, the Intelligence or Internal Organizing System, has been added to reflect an understanding of the role of information in the decision-making process. Organizational actions which are the outputs of the efferent (decision) subsystem are not based upon outputs of the afferent (scanning) subsystem, but rather upon the outputs of the intelligence (internal organizing) subsystem which acts as an evaluator of the receptor or scanning subsystem. Raw sensory data received by the scanning (receptor) system is eventually fed into the decision (efferent) system when it is utilized for problem-solving purposes (Ref 49).

The next step of this sequence is a selective conversion and organization of the data into a form suitable for consumption by management. It is only through this conversion process that the data can become information. It is this information which serves as a basis for decision-making (Ref 49).

It must be emphasized that none of the three subsystems

(scanning, organizing, decision) shown in Figure 8 operate independently or constitutes a separate unit within the organization. It is quite likely that one or several individuals within the system development organization may be engaged in any or all three of the activities at various times.

This interaction system has been treated conceptually as a hierarchical system. A hierarchical system is one that is composed of interrelated subsystems, each subordinate to the one above it (Ref 53). This "boxes-within-boxes" way of looking at a complex phenomena is not a mere partitioning of elements but one joined to the relationships of the several parts with one another and with the whole. This interaction is regulated by the decision-making process (Ref 49).

Moreover, the actions of the software system development organization, shown in Figure 8, are iterative in nature because for each phase of the SDLC another iteration (or flow through the three subsystems) occurs.

In other words, to advance from one phase of the SDLC to the next, a decision has to be made. This decision determines if the current phase is complete and that the development effort should continue to the next phase, or resolves problems in that current phase.

The procedures for defining software-oriented quality attributes, which were presented in Chapter II, are used within this model to determine what data is appropriate and

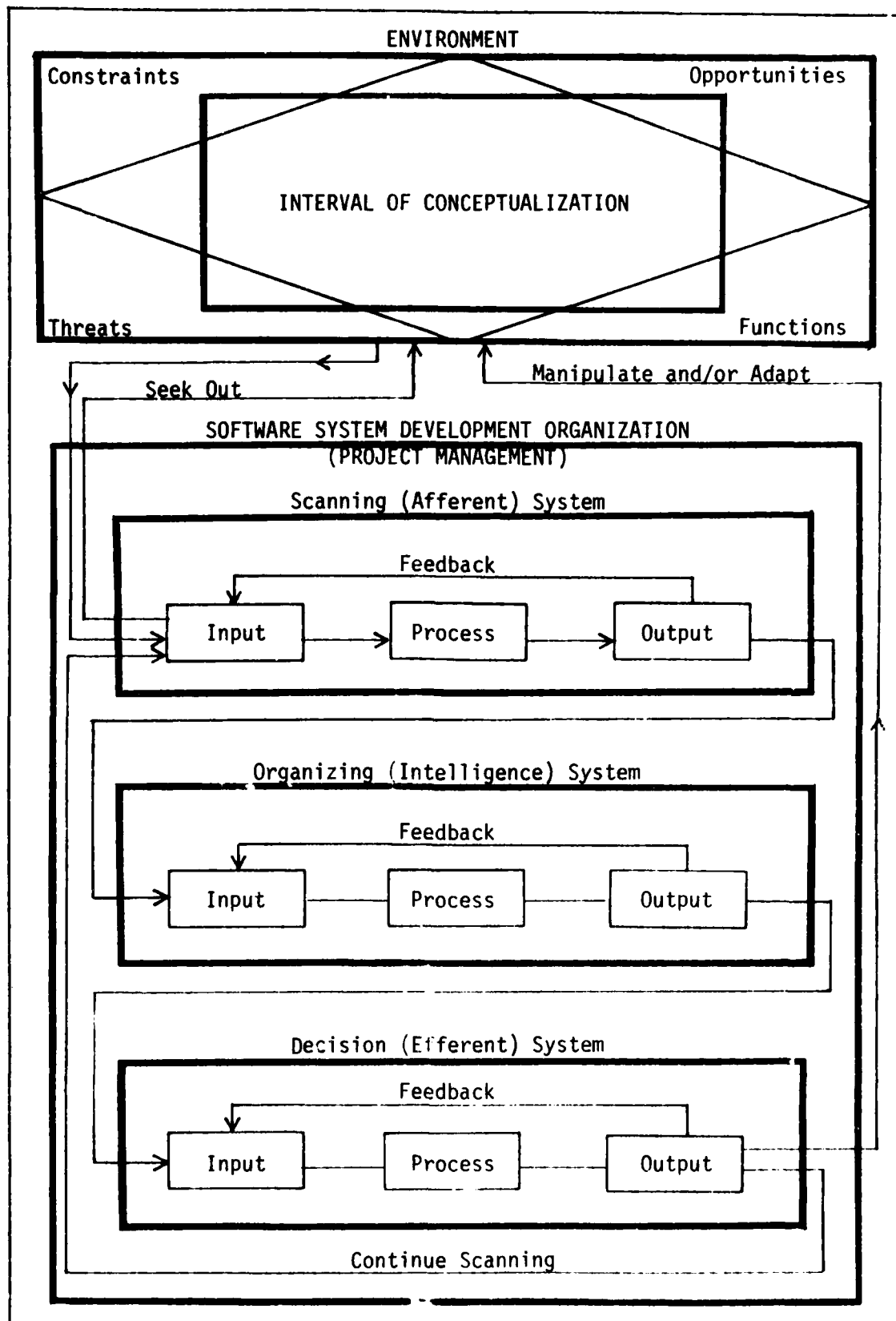


Figure 8. Software System Development - Environment Interaction System

to convert the data into usable information, which is, in turn, used by the software system development organization to make decisions about the progress of the development effort.

Imposing Metric Framework on the Model

The first step in applying the framework for software quality is to conceptualize the environment in which the user's system will be operational. It is important to realize that individual managers within the user group will have different views or conceptualizations of the user environment (Ref 58). Each manager has a different concept of what the constraints, threats, opportunities and functions of the user environment are. Moreover, this view of the user environment is constantly changing with time because of changes in requirements, regulations, etc. Therefore, it is important to establish a group consensus of the user environment for a specific period or interval of time. In practice, this "interval of conceptualization" will be the top executive's model of his user organization (hopefully with input from lower level managers).

This conceptualization of the user environment, establishes the desired or required characteristics of the proposed software system which must support the user operations.

Table IV in Chapter II, page 33, provides a basic checklist to help managers identify these software system characteristics. Figure 9 illustrates an example of

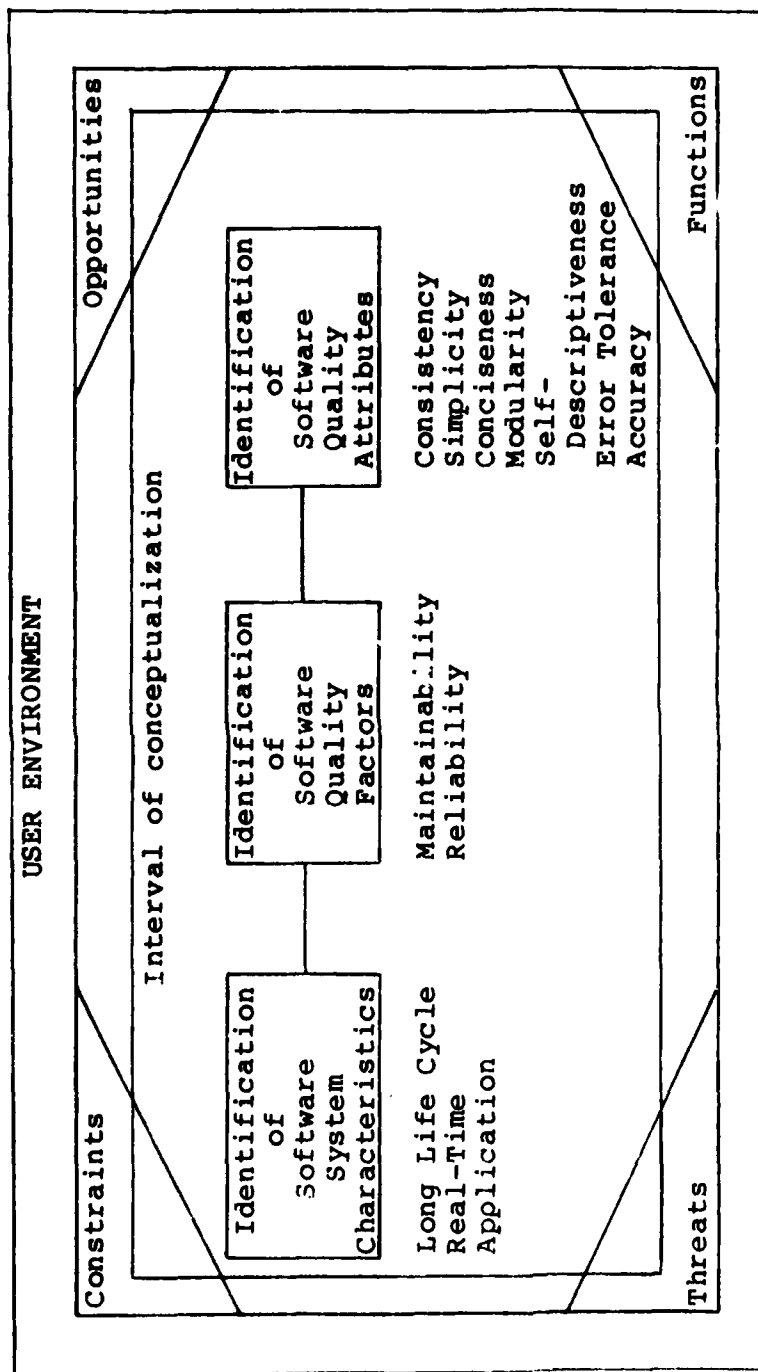


Figure 9. Defining the User's System Requirements

applying this software metric methodology.

By establishing the important characteristics of the system as being Real Time Application and having a Long Life Cycle, a manager (or management group) can then use Table III in Chapter II, page 32, to more easily isolate the software quality factors which are related to the system characteristics.

By using Tables III and IV in interviews with AFLC/LM personnel, six software quality factors initially emerged as candidate choices. From the system characteristic of a Real Time Application, three factors were related: Reliability, Efficiency, and Correctness. Having a Long Life Cycle as the other system characteristic produced three more factors: Maintainability, Flexibility, and Portability.

Table V in Chapter II, page 34, was used to tailor these factors to AFLC's cost (and environmental) constraints. The table shows that the factors can be grouped according to system life cycle activities associated with a delivered software product. The cost-to-implement versus life-cycle-cost-reduction relationship exists for each quality factor. It was this relationship and the life cycle implications that made Reliability and Maintainability the two critical factors for this system, G072A.

Tables VI and VII in Chapter II on pages 35 and 36, further substantiated this selection of Reliability and

Maintainability in that the relationship between these two factors was synergistic.

Ratings for the two quality factors could be established using Table XIX. For example, a reliability rating of .99 requires less than one error for every 100 lines of code to be detected during formal testing. A maintainability rating of .8 requires two man-days as an average level of maintenance for correcting an error. These ratings can also be established at each measurement period during the development as follows:

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

The progressively better scores are required because there is more detailed information in the later phases of the development to which to apply the metrics and more confidence in the metrics' indication of quality. This is analogous to the concept of reliability growth (Ref 28).

The next step in establishing the software quality requirements specification used Table VIII in Chapter II on page 38 to proceed from the management-oriented quality factors to the software-oriented criteria. By design, the identification of these criteria was automatic and represented a more detailed specification of the quality requirements.

The software criteria which were established from Reliability were: Error Tolerance, Consistency, Accuracy,

Table XIX

Quality Factor Ratings

QUALITY FACTOR	RATING EXPLANATION	RATING GUIDELINES				
		RATING	.9	.98	.99	.999
RELIABILITY	Rating is in terms of the number of errors that occur after the start of formal testing. Rating = $1 - \frac{\text{Number of Errors}}{\text{Number of Lines of source code excluding comments}}$	ERRORS				
		TOO LOC	10	2	1	.1
MAINTAINABILITY	Rating is in terms of the average amount of effort required to locate and fix an error in an operational program. Rating = $1 - .1$ (Average number of man days per fix)	RATING	.3	.5	.7	.9
		AVERAGE EFFORT (MAN DAYS)	7	5	3	1

and Simplicity. The criteria established from Maintainability were: Consistency, Simplicity, Conciseness, Modularity, and Self-Descriptiveness. The definitions of these factors and criteria are provided in Appendix E. These definitions, which are summarized in Table XX, along with an explanation of the metrics for each of the criteria (described in Appendix C) should be provided to systems analysts, programmers and anyone else involved in the software development process.

An important management concept was realized at this initial phase. Gerald Weinberg demonstrated that goal-directed system development did much to improve the quality of the software system (Ref 27). By stating up-front the specific attributes which the system must exhibit, project management can greatly influence the effectiveness of the end-product.

Furthermore, the structured methodology of the framework for Software Quality Metrics enables managers, who know little about software but much about what is needed to support the user group, to define their requirements specifications in software-oriented terms. At the beginning of the conceptual phase, software-oriented requirements can be specified in the Data Automation Requirement (DAR) and the Data Project Directive (DPD).

Monitoring the Development Effort

The framework for Software Quality Metrics provides the quality assessment capability needed by management in its

AD-A115 501

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/8 9/2
SOFTWARE QUALITY METRICS: A SOFTWARE MANAGEMENT MONITORING METH--ETC(U)
MAR 82 S J JARZOMBEX
AFIT/SCS/MA/SEN-1

UNCLASSIFIED

NL

2-4
Sh:101

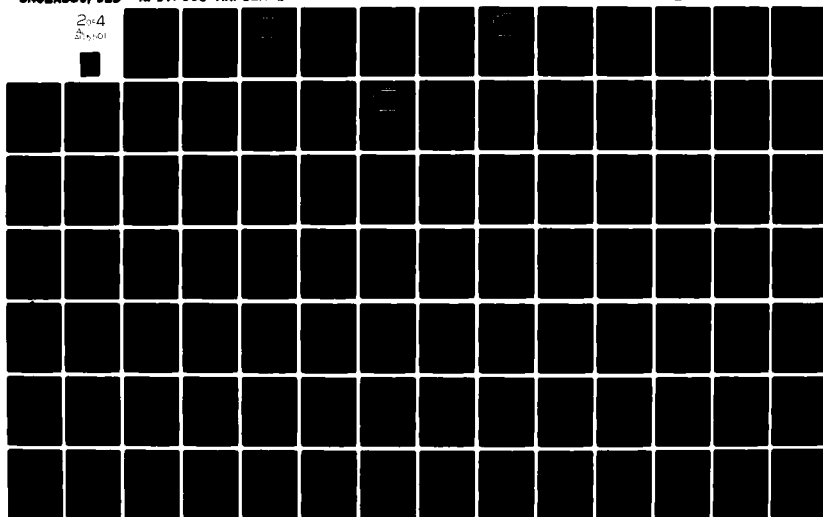


Table XX

Definitions Relating to Reliability and Maintainability

Definition of R&M Factors		
<u>RELIABILITY</u>	Extent to which a program can be expected to perform its intended function with required precision.	
<u>MAINTAINABILITY</u>	Effort required to locate and fix an error in an operational program.	

Criteria Definitions		
CRITERION	DEFINITION	RELATED FACTORS
CONSISTENCY	Those attributes of the software that provide uniform design and implementation techniques and notation.	Reliability Maintainability
ACCURACY	Those attributes of the software that provide the required precision in calculations and outputs.	Reliability
ERROR TOLERANCE	Those attributes of the software that provide continuity of operation under nonnominal conditions.	Reliability
SIMPLICITY	Those attributes of the software that provide implementation of functions in the most understandable manner. (Usually avoidance of practices which increase complexity.)	Reliability Maintainability
MODULARITY	Those attributes of the software that provide a structure of highly independent modules.	Maintainability
SELF-DESCRIPTIVENESS	Those attributes of the software that provide explanation of the implementation of a function.	Maintainability
CONCISENESS	Those attributes of the software that provide for implementation of a function with a minimum amount of code.	Maintainability

decision-making process. Figure 10 superimposes the application of Software Quality Metrics onto the decision-making process. It represents the conceptual structure of the decisioning process for one phase of the SDLC. It is within this iterative structure that the lowest level of the Software Quality Metrics framework -- the metrics -- is applied. This level enables the quantification of software quality.

Displayed in Figure 9, the software quality attributes, established in conceptualizing the system requirements of the user environment, require system level and module level collection of data which is to be collected during the phases of the SDLC. The specific data to be collected is requested on metric worksheets listed in Appendix A.

A Conceptual Walk-Through

Currently, the volumes of documentation generated during the SDLC under Configuration Management serve to overwhelm project management. Indeed, the magnitude of data cannot be "consumed" in problem-definition or in decision-making.

The scanning (afferent) system must therefore be selective in its collection of data, and should have the capability to ascertain what data is required for decisioning. This is acquired through the use of Software Quality Metrics. To promote an understanding of how the metrics can provide the assessment of software quality, a conceptual walk-through of Figure 10 must be performed.

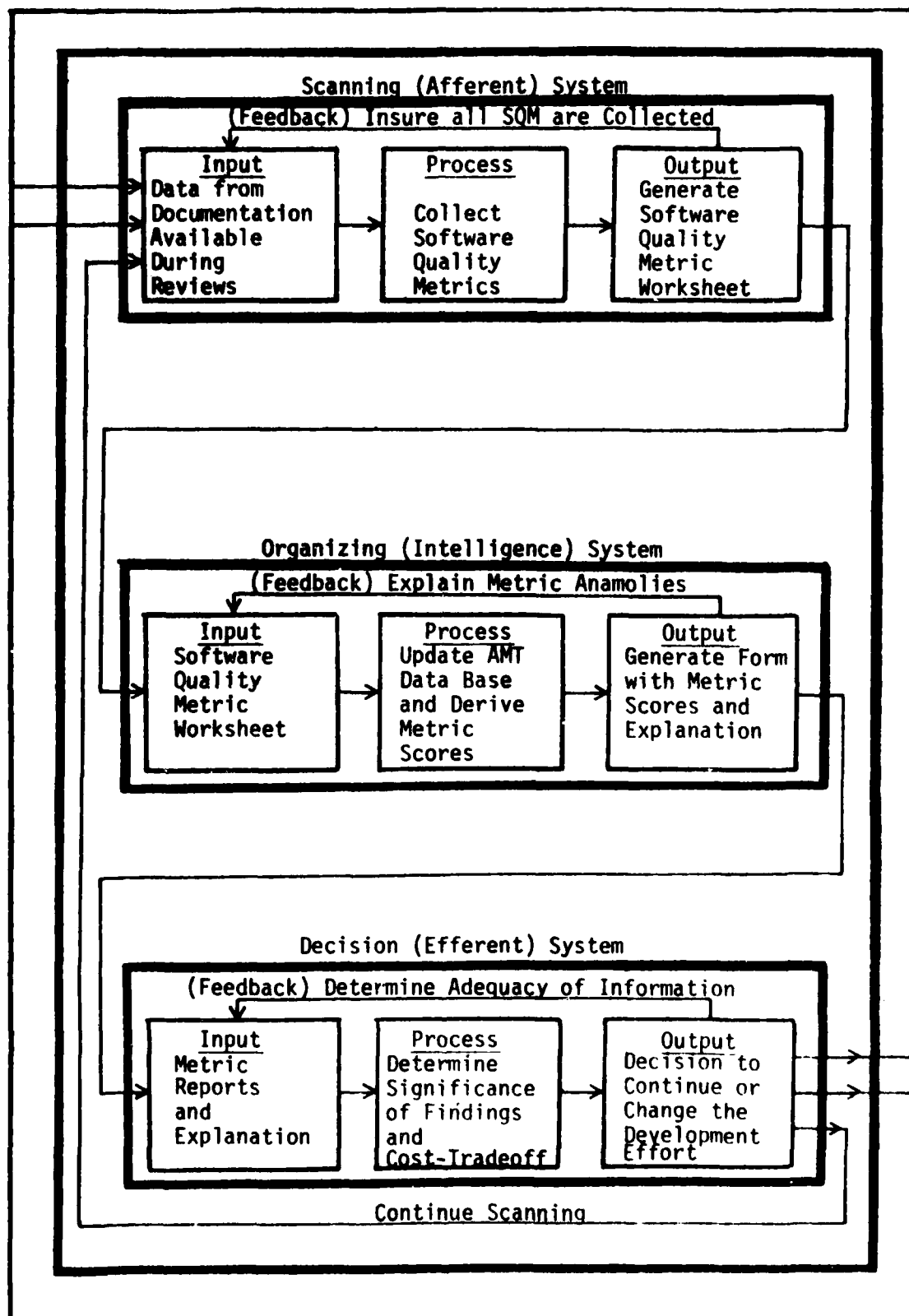


Figure 10. SQM Applied to the Decision-Making Process

Input to the scanning system is the raw sensory data generated by software development documentation, source code, etc. The metric worksheets provide the selection process by specifying what data to obtain. The up-front specification of software factors and attributes has narrowed the scope of data to be collected. Specifically, by determining Reliability and Maintainability as the software factors, metric data is collected which assesses the software attributes: Consistency, Simplicity, Conciseness, Modularity, Self-Descriptiveness, Error Tolerance, and Accuracy. Though other metrics should also be collected, management attention can be focusing on the achievement of Reliability and Maintainability.

Moreover, the metric worksheets serve as feedback to insure that documentation is complete, i.e., confirming that the information that is requested by the worksheets is actually found in the system documentation. This should warn project management that documentation is incomplete, thus allowing early detection of a problem. Also serving as a checklist to insure that all metric data is collected, the worksheets function as the output of the scanning system and as the input to the organizing (intelligence) system.

The process internal to the organizing system is the actual update of the Automated Measurement Tool (AMT) data base. In turn the AMT automatically computes the metric scores by using the algorithms listed in Appendix D. The

AMT provides a report generation function, which was discussed in Chapter IV, that can be used to provide specific information to project management, the system analysts and programmers, the SQA staff, the test group, and researchers. The nine reports it generates (Worksheet, Summary, Matrix, Statistics, Quality Growth, Normalization, Module, and Exception Reports) can be complemented by organizational forms specific for AFLC/LM.

These forms provide the basis for feedback in resolving inconsistencies in actual metric scores to what may have been expected. This provides direction to the development staff to look in specific areas of the programs for possible problems. As output of the organizing system, these forms provide the input to the decision (efferent) system.

At this point, management is in a better-informed state to make decisions about the quality of the system. The forms and explanations provide the decision system with information (not raw data) about possible cost trade-offs, various development alternatives, and the quality of the software. The feedback would be to determine the specific information that could further enhance the decision-making process.

The output is the decision: to continue on to the next phase, to repeat portions of the current phase or to fall-back onto a portion of a previous phase to change a part of the development effort. For example, if an

opportunity could be realized by changing a requirement, even though the development effort is in the design phase, then some of the procedures would have to be duplicated and a pass through the three systems (scanning, organizing, and decision) would insure completeness of the change.

It is anticipated that by accomplishing the structured-flow through the scanning, organizing and decision systems and by satisfying the information requirements of the Software Quality Metrics, that the appropriate decision at the end of each phase will be to continue to the next phase.

Metric Application Throughout the SDLC

Each phase of the System Development Life Cycle is an iteration of Figure 10; however, there is a sufficient amount of variation among the phases to warrant a discussion of the flow through the entire SDLC. The entire structural model of the SDLC superimposed by the framework for Software Quality Metrics is displayed at the end of this chapter in Figure 15.

The procedures for describing the software-oriented attributes of the system, which are depicted in Figure 9, remain the same for each system development.

Requirements Analysis Phase. Represented in Figure 11, the Requirements Analysis is the first phase of the SDLC and is called the Conceptual Phase under Configuration Management at AFLC/LM. At the end of this phase, the

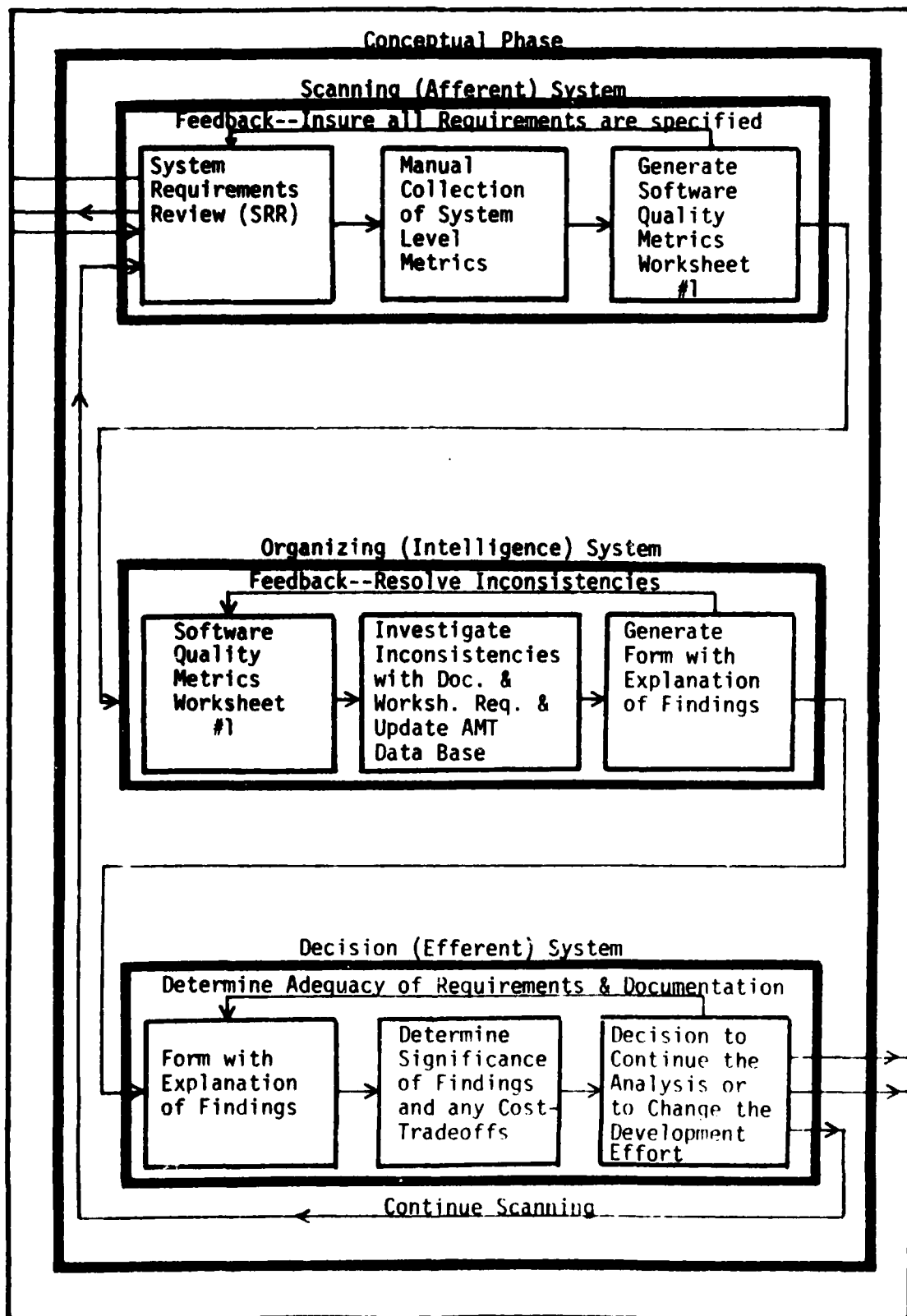


Figure 11. Requirements Analysis

decision-making process is extremely important because the Software Requirements Review (SRR), which establishes the Functional Baseline, serves as the milestone that also represents the beginning of the Definition Phase. Therefore, the data which serves as the basis for the decision-making process plays a key role in determining the completeness of the Requirements Analysis.

The primary input to the scanning system during the Requirements Analysis is the documentation specified for the SRR under Configuration Management. Other documents, such as the Data Automation Requirements (DAR) and Data Project Directive (DPD) also serve as sources for input data. The scanning process is the manual collection of system level metrics as specified by Metric Worksheet #1 in Appendix A.

Because it was decided up-front that the software system must exhibit the quality factors of Reliability and Maintainability, the data collected, which should be available during the systems requirements analysis, is information which checks:

- if error analysis has been performed and budgeted to functions;
- if there are definitive statements of the accuracy requirements for inputs, output, processing and constants;
- if there are definitive statements of the error tolerance of input data;
- if there are definitive statements of the requirements for recovery from computational failures;
- if there is a definitive statement of the requirements for recovery from hardware faults;
- if there is a definitive statement of the requirements for recovery from device errors;
- the number of major functions and data references; etc.

The Software Quality Metric Worksheet #1 provides the feedback to project management and the SQA function by providing a checklist to insure that a complete analysis of requirements has been conceptualized and documented. The Metric Worksheet #1 functions as the output document of the scanning system and as the input document of the organizing system within the Requirements Analysis Phase of the SDLC.

The process within the organizing (intelligence) system updates the Automated Measurement Tool (AMT) data base and investigates inconsistencies within documentation, requirements analysis, and worksheet specifications. For example, certain requirements of the system may be in conflict with the attributes which seek to insure that the system will be Reliable and Maintainable. These findings can be discussed on an organizational form which explains these possible inconsistencies.

This organizational form serves as the primary output of the organizing system and the input for the decision system. Although the metrics reports can be generated by the AMT, they provide very little useful information this early in the SDLC (during requirements analysis). The process involved within the decision system is to determine the significance of the findings generated by the organizing system. Cost trade-offs should be considered in keeping the requirements as they are, or to modify the system requirements to insure high Reliability and Maintainability. It is the decision system which provides

the feedback to determine the adequacy of the system requirements and documentation.

By utilizing the feedback within and between the subsystems and applying the metric concepts, it is anticipated that the logical output of the decision system will be to continue to the Design Phase of the SDLC.

Design Phase. Represented in Figure 12, the Design Phase is the second phase of the SDLC, and it is the equivalent of the Definition through Detail Design Phase under Configuration Management for AFLC/LM. This phase plays an important role in the SDLC decisioning process because the System Design Review (SDR) which establishes the Allocated Baseline, the Preliminary Design Review (PDR), and the Critical Design Review (CDR) all serve as decision milestones. Therefore, the data, that is made available on Design Phase documentation and serves as the basis for the decision-making process, plays a key role in determining the completeness of the Design Phase.

Each review (SDR, PDR, and CDR) represents another iteration of the flow through this phase. In the true sense, this is a recursive phase -- it repeats itself in the collection of data during each design review. Therefore, the input to the scanning system during the Design Phase depends upon the particular review or iteration of this phase.

The scanning process is the manual collection of both system level and module level metrics as specified by

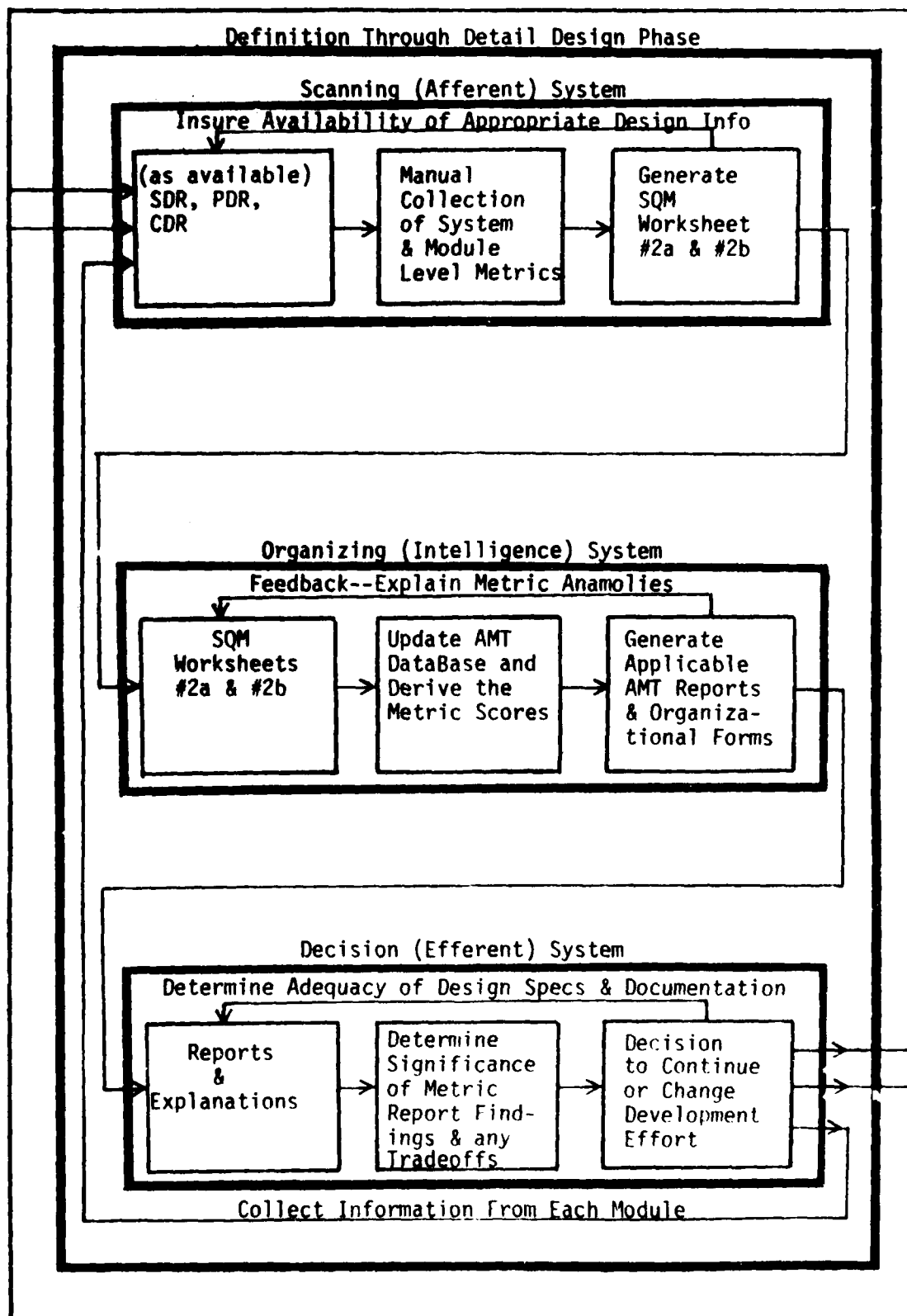


Figure 12. Design Phase

Metric Worksheets #2a and #2b which are listed in Appendix A. It should be noted that if a standard design language such as PDL (Program Design Language) is used, then the AMT can be used to collect the data. The efficiency of this automatic collection of data not only saves time and human resources but comes close to providing real-time feedback on the computation of metric scores.

Because it was decided up-front that the software system should be highly reliable and maintainable, specific data, which should be available during the design phase, is collected at both the system level and the module level, as indicated by the metric worksheets.

The data which the Metric Worksheet #2a collects is system level information which checks:

- if match library routines to be used have been checked for sufficiency with regards to accuracy requirements;
- if concurrent processing is centrally controlled;
- if error conditions are reported by the system;
- if errors are automatically fixed or bypassed and if processing continues;
- if errors require operator intervention;
- if provisions for recovery from hardware faults and device errors are provided;
- if a hierarchy of system, identifying all modules in the system, is provided;
- if the number of modules is recorded; if so, what is the number of modules.

The data which Metric Worksheet #2b collects is module level information (collected from each module) which checks if:

- numerical techniques being used in algorithms have been analyzed with regard to accuracy requirements;
- values of inputs range have been tested;

conflicted requests and illegal combinations have been identified and checked;
there is a check to see if all necessary data is available before processing begins;
all input is checked, reporting all errors before processing begins;
loop and multiple transfer index parameters range are tested before use;
subscripts range are tested before use;
outputs are checked for reasonableness before processing continues;
etc.

The Design Phase metric worksheets provide feedback to project management and the SQA function by providing a checklist to insure that the design specifications adequately describe how the system requirements will be achieved. Moreover, the worksheets insure that the necessary design information is documented. Worksheets #2a and #2b function as the output documentation of the scanning system and as the input documentation of the organizing system within the Design Phase of the SDLC.

The process within the organizing (intelligence) system updates the AMT data base and derives metric scores. It also seeks to explain metric anomalies -- inconsistencies with historical data versus current metric scores. The reports that are generated by the AMT provide useful information even in this early phase. Organizational forms should be generated to document any inconsistencies and to provide explanations of the metric reports. These forms will be used with the metric reports as output of the organizing system and as input to the decision system. The decision system determines the significance of the metric

reports and organizational reports. Again trade-offs must be considered, and a determination of the adequacy of design specifications and documentation can be challenged. By using the feedback within and between the subsystem and applying the metric concepts, it is anticipated that the logical output of the decision system will be to continue to the Implementation (coding & checkout) Phase.

Implementation Phase. Represented in Figure 13, the Coding and Checkout Phase is the third phase in the SDLC. This phase is the equivalent of the Development (Code through Subsystem Test) Phase under Configuration Management for AFLC/LM.

Detailed Design Specifications from documentation which resulted from the CDR serve as the preliminary data input to the scanning system. Once programming has begun, the source code provides data input. During the subsystem or unit test, the Preliminary Functional/Physical Configuration Audits (FCA/PCA) and Product Verification Review (PVR) represent key decision-making points during the SDLC. As a result of the PVR, the Product Baseline is established. Therefore, the input to the scanning system during the Implementation Phase depends upon the particular iteration (review, audit or code) of this phase.

The scanning process is the manual and AMT collect of module level metrics as specified by Metric Worksheets #3 and #2b which are listed in Appendix A. If a standard design language is used, the AMT can be used to

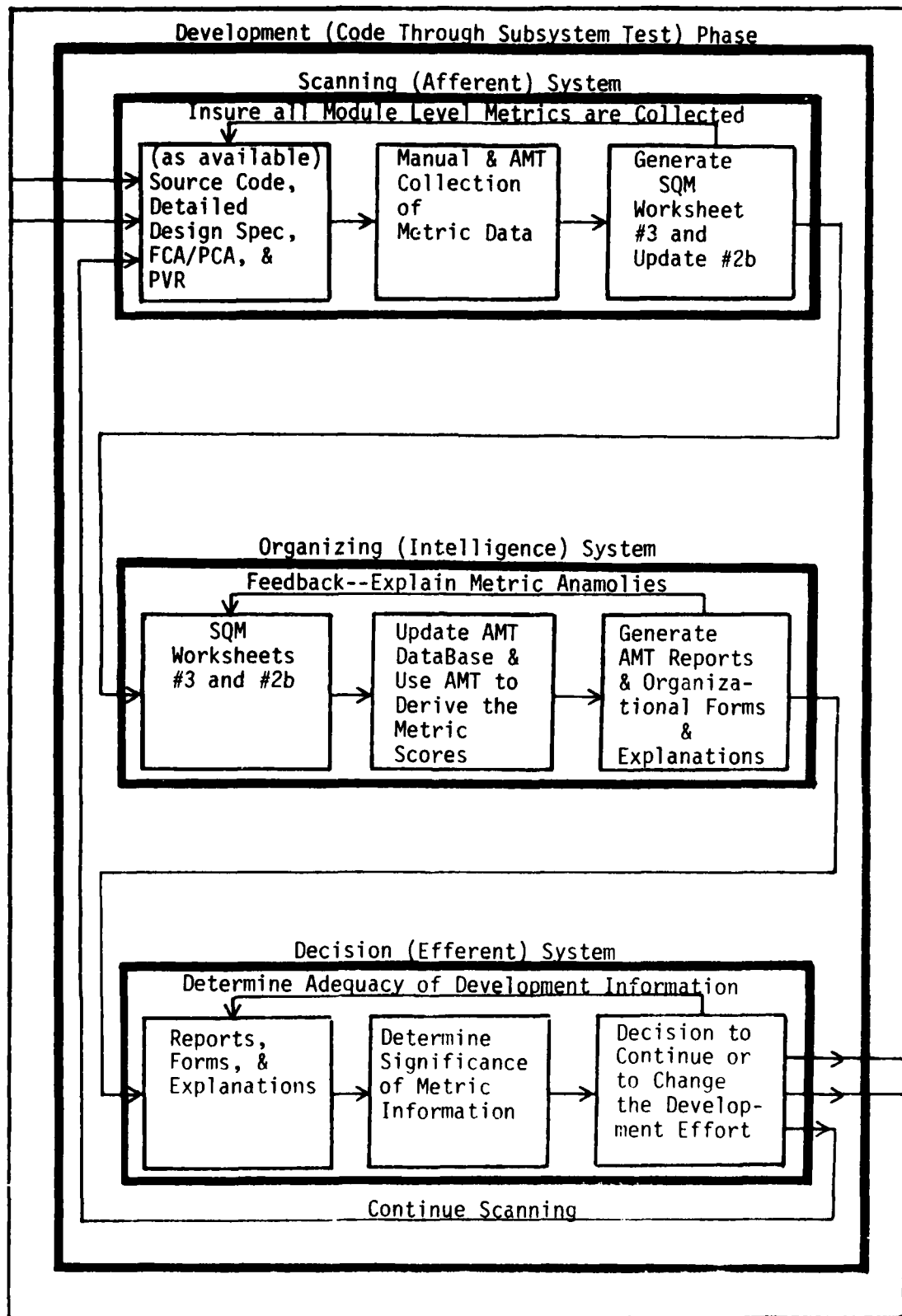


Figure 13. Programming & Checkout

automatically collect the detailed design data. The AMT automatically collects source code data for module level metrics. A metric worksheet is used for every module to insure all metric data is collected. As a module is changed, Worksheet #2b must also be updated, thus replacing the previous Worksheet #2b for that module.

To insure that software exhibits Reliability and Maintainability, specific data must be present. The metric data which Worksheet #3 collects from each module checks:

- the number of decision points;
- the number of declarative statements;
- the number of lines excluding comments;
- the number of statement labels;
- the number of conditional branches;
- the number of unconditional branches;
- the number of loops;
- the number of lines of comments;
- if there are prologue comments containing information about the function, author, version number, date, inputs, outputs, assumptions and limitations;
- the number of decision points and transfers of control that are not commented;
- if all machine language code is commented;
- etc.

The data on Metric Worksheet #2b needs to be updated as the modules are changed checks if:

- numerical techniques being used in algorithms have been analyzed with regard to accuracy requirements;
- values of inputs range have been tested;
- conflicted requests and illegal combinations have been identified and checked;
- there is a check to see if all necessary data is available before processing begins;
- all input is checked, reporting all errors before processing begins;
- loop and multiple transfer index parameters range are tested before use;
- subscripts range are tested before use;
- outputs are checked for reasonableness before processing continues;
- etc.

The Implementation Phase metric worksheets provide feedback to project management and the SQA Function by providing a checklist to insure that the coding and checkout procedures adequately produce the specified software system. Moreover, the worksheets insure that the necessary development information is documented. Worksheets #2a and #3 for each module function as the output documentation of the scanning system and as the input documentation to the organizing system within the Implementation Phase of the SDLC.

The process within the organizing (intelligence) system updates the AMT data base, derives metric scores, and seeks to explain metric anomalies. The reports that are generated by the AMT provide a quantified assessment of software quality. Organizational forms should be generated to document any inconsistencies and to provide explanations for the metric scores. These forms will then be used with the metric reports as the output of the organizing system and as the input to the decision system of the Implementation Phase.

The decision system determines the significance of the metric reports and organizational forms. At this point an adequacy of development documentation can be challenged -- the metric reports will show the problem areas. Management can inquire of the amount of code coverage (number of paths executed) and determine if all code can be reached. By using the feedback within and between the subsystems and

applying the metric concepts, it is anticipated that the logical output of the decision system will be a smooth transition into the Test-and-Integration Phase.

Test and Integration Phase. Represented in Figure 14, the Test-and-Integration is the fourth phase in SDLC. This phase is the equivalent of the test phase (and System Test in the Development Plan) under Configuration Management for AFLC/LM.

The subsystem/system test results and PVR serve as the preliminary input to the scanning system. The documentation associated with the final FCA/PCA is also available during the initial part of the test phase. Finally the System Validation Review (SVR), which represents a key decision-making point of the SDLC, marks the end of the test phase. Therefore, the documentation associated with the SVR must detail all the data which is necessary to insure a reliable and maintainable software system. Because this becomes available at different points during the test phase, the input to the scanning system during the test phase depends upon the particular iteration (flow through) of this phase.

The scanning process can rely mainly on the AMT to collect the module level metrics as specified by Metric Worksheet #3, which is updated from the implementation phase. Because this phase is testing for the total system integration, it is necessary to update Metric Worksheet #2a to insure that any changes to modules during the

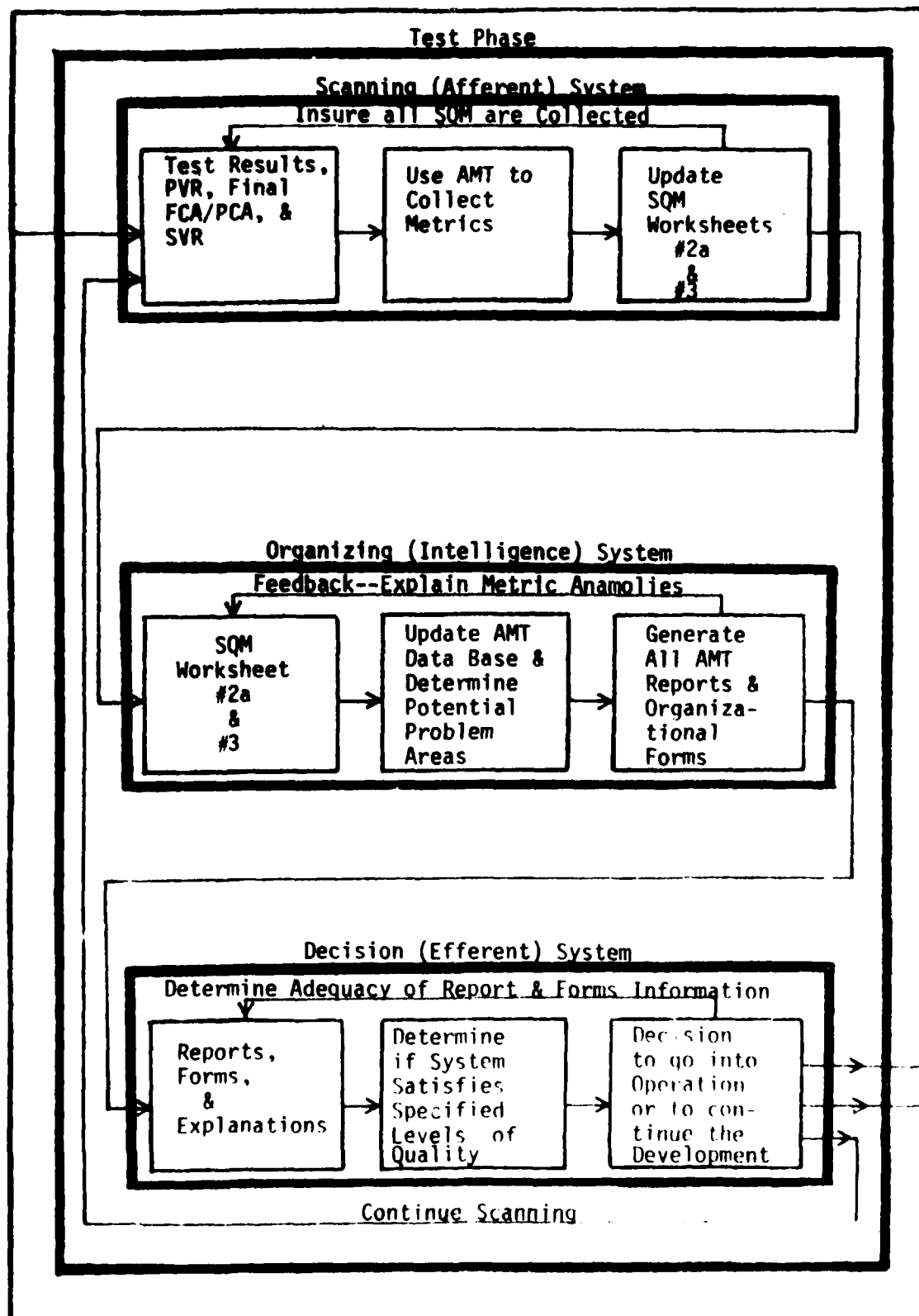


Figure 14. Test and Integration

Implementation Phase did not adversely affect the system level performance. The updates to the metric worksheets provide the checklist to insure necessary documentation and feedback to project management and the SQA function. The metric worksheets, #3 for each module and #2a for the system, function as the output documentation of the scanning system and as the input documentation to the organizing system within the Test-and-Integration phase of the SDLC.

The process within the organizing (intelligence) system updates the AMT data base, derives metric scores, and determines potential problem areas (modules). The reports that are generated by the AMT provide a quantified assessment of the software system quality. Organizational forms should document the potential problems within modules identified by the metric reports. These forms will then be used with the metric reports as the output of the organizing system and as the input to the decision system of this phase.

The decision system determines if the system satisfies the specified levels of quality, as stated during the requirements analysis. The decision-makers must decide if the potential problem areas should be investigated or to simply identify the modules and allow the system to go into operation. By using the feedback within and between the subsystems and applying the metric concepts, it is anticipated that the logical output of the decision system

will be to allow the system to become operational.

Figure 15 shows the interrelationship between the phases by indicating the possible flows through the SDLC. Because each phase has already been depicted in Figures 11 through 14, the legibility of each element is not important. The significance of Figure 15 is to demonstrate that the entire SDLC is iterative: processes are repeated within phases, phases can be repeated, and decisions can be made to "fall back" to any previous phase. It is necessary to understand that changes introduced later in the life cycle require a flow through the entire model. Only then can the developer insure a "complete" software system.

Chapter Summary

Conceptualized within the context of the decision-making process, software development has been portrayed as an iteration of subsystems within the various phases of the SDLC that require specific information to enhance decisioning. The utilization of the SQM concepts provides a quantitative assessment of software quality for a goal-directed system development which is controlled by management.

Chapter VI develops the checklists which enhance the SQM concepts by providing prescriptive elements which are indicators of reliable and maintainable software.

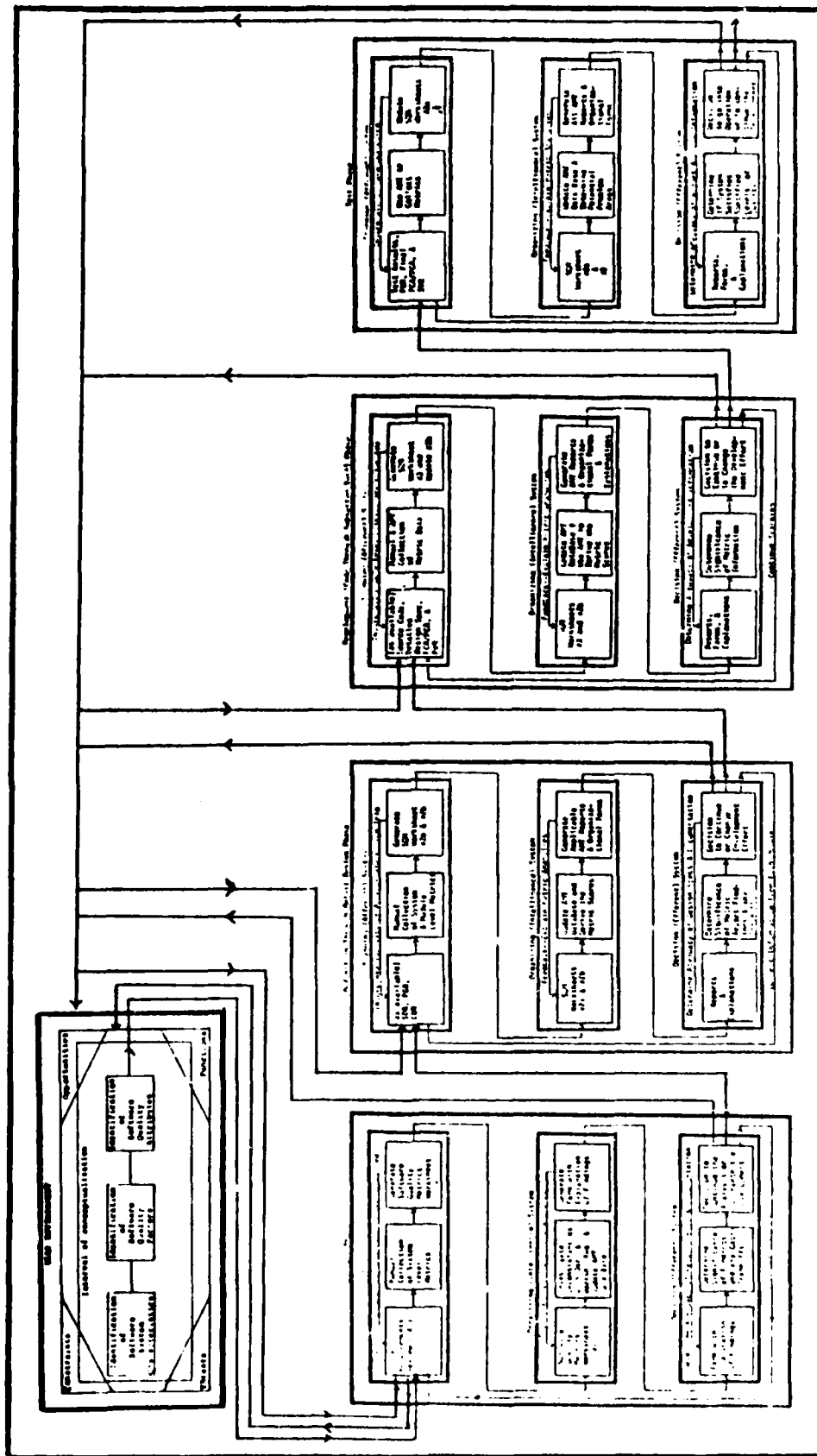


Figure 15. The Mapping of SQM to the SMLC

Copy available to DTIC does not
 permit fully legible reproduction

CHAPTER VI

SQA INFORMATION REQUIREMENTS: A CHECKLIST

To insure a quality software system, the deliverable products must provide information to demonstrate that the software system exhibits the desired quality attributes. The metrics, which establish the quantitative assessment of that software quality, are not based on the availability of specific documentation, but rather on the availability of key information. With this in mind, it is the intent of this chapter to satisfy the third goal of this thesis: to demonstrate how the SQM concepts can be applied at AFLC. This is accomplished by providing checklists for system developers. The checklists help to insure that key information is made available during the SDLC. Because the checklists use items from the SQM worksheets, which are descriptive in nature, they expand the SQM concepts by providing a more prescriptive means for developing quality software.

Documentation Requirements

One of the main considerations in applying Software Quality Metrics is the availability of data or software products which provide the sources for collection of the

metrics. Software products include the source code (the most obvious and researched source of metrics to date), documentation including requirements specifications, design specifications, manuals, test plans, problem reports, correction reports, and reviews (Ref 24).

There are basic documentation requirements regardless of the size or cost of a program. Certainly, less documentation is produced during a low cost/short life project, but it should still contain certain information. The documents may not be in as much detail, yet they still must contribute to the quality of the software product. Indeed, this is the situation for AFLC. The program classification dictates the documentation requirements. Major programs are better documented because of upper management attention. For example, MMSIP (Maintenance Management Systems Improvement Project) and its subsystems, received upper management's attention because they provide cost information about organic depot maintenance to aid in managing the Depot Maintenance Services Branch of the Air Force Industrial Fund (DMS, AFIF). Because of this attention, G072A MMSIP, which is a subsystem of MMSIP, is supposed to be one of the better documented systems at AFLC (Ref 51).

However, it was difficult to extract various data/information because the documentation either did not provide the information, or did not clearly identify the data. Now this point is critical in considering

establishing Software Quality Metrics and the Automated Measurement Tool. Though the format was in compliance with regulations, the availability of key information was not made readily apparent.

To insure a quality product, the documentation of that software must provide key information related to the desired quality attributes. It is an organizational decision to determine the placement of the information into specific documents.

To aid developers in documenting a software system's reliability and maintainability, a checklist of standards has been developed for each phase of the SDLC. These checklists provide goal-directed documentation procedures because developers are aware of the specific information requirements, not just format.

Application for Checklists with SQM

If the development team were to use the checklists in this chapter with existing Question Sets listed in AFLC's Project Manager's Handbook (Working Copy), then it would be combining software-oriented requirements of the checklists with the user or management-oriented requirements of the Question Sets. To demonstrate the uses of the checklists and SQM for the G072A MMSIP, one must consider what would have happened. This is easiest to visualize by applying the model developed in Chapter V to illustrate deficiencies in current development practices. Because the AMT is not yet available to AFLC, it was infeasible to collect module

level information. Therefore, it is only possible to illustrate the first two phases of the SDLC.

Conceptual Phase. At the beginning of the Conceptual Phase several documents were generated to reflect the requirements of the user. The Environmental Assessment/Statement accompanied the Data Automation Requirement (DAR) through the decision-making process to HQ USAF. It was during this initial phase that the end-user identified the system characteristics. It was known that the system would have a long life cycle and that real time application was needed. Because maintainability had become a command level requirement (Ref 89), it was easy to identify it as a required software factor. However, there was no further definition of software attributes.

By using the step-wise development from system characteristics to software attributes, depicted earlier in Figure 9 on page 82, AFLC's project management could have further definitized system requirements. Moreover, it could have specified a Reliability rating and a Maintainability rating as shown in Table XIX on page 85. This could have provided up-front criteria on which to base final acceptance of the system during the testing phase.

This failure to provide software-oriented requirements meant that, at the time of the Systems Requirements Review (SRR), the system developers were unable to assess the system's reliability and maintainability. If the SQM

worksheets would have been applied at that time, only the items which corresponded to Computer Program Configuration Items (CPCI) would have been satisfied. For example, the description of the flow of processing, and all decision points in that flow, was provided under CM, however, items relating to error tolerance were not even addressed; so there was a gap in current procedures under CM.

This gathering of data would complete the scanning phase (system) of the Requirements Analysis shown in Figure 11 on page 93. The immediate need for a checklist can now be seen as a way to resolve the inconsistencies characterizing current documentation requirements under CM to that of a more desired development effort.

Therefore, to enhance the Question Sets in the Project Manager's Handbook and the procedures under CM, a checklist has been developed that will make system designers aware of items that are present in reliable and maintainable systems during the Conceptual Phase. At the beginning of the Requirements Analysis, system designers should be given this checklist to insure that:

1. Requirements are itemized so that the various functions to be performed (their input and outputs) are clearly delineated;
2. All data references are defined;
3. All defined functions are justified;
4. All referenced functions are defined;
5. All data referenced are used;
6. A description is provided for the flow of processing and all decision points in that flow;

7. All problem reports, related to the requirements are recorded and closed (resolved) prior to the System Requirements Review (SRR);
8. An error analysis has been performed and budgeted to functions;
9. There are definitive statements about the accuracy requirements for inputs, outputs, processing, and constants;
10. There are definitive statements of the error tolerance of input data;
11. There are definitive statements of the requirements for recovery from: (1) computational failures, (2) hardware faults, and (3) device errors.

Only after this additional data has been gathered can the organizing system provide adequate information to the decision system. Without this additional data, a decision should not have been made to continue on to the next phase of development, for the Requirements Analysis was truly incomplete.

Design Phase. However, the development effort did continue into the Design (Definition through Detail Design) Phase. Again, the only data that could be gathered on the worksheets by the scanning system was that which corresponded to CPCI's.

The following checklist has been developed to fill this gap of information requirements. This information should appear in the draft of the user's manual. Prior to beginning the Design Phase, system developers should insure that:

1. There is a matrix relating itemized requirements to modules which implement those requirements;
2. All major functions are identified, defined, and used;
3. All interfaces between functions are defined;
4. All problem reports are resolved;
5. A profile of the number of problem reports is "broken-out" by the following types--
Computational, Logic, I/O, Data Handling, OS/System Support, Configuration, Routine/Routine Interface, Tape Processing, User interface, Data base interface, User requested changes, Preset data, Global variable definition, Recurrent errors, Documentation, Requirement compliance, Operator, Questions, Hardware;
6. Math library routines to be used have been checked for sufficiency with regard to accuracy requirements;
7. Concurrent processing is centrally controlled;
8. All error conditions are reported by the system, and determine how many of these errors are automatically fixed or bypassed and allow processing to continue, and how many require operator intervention;
9. Provisions for recovery from hardware faults and device errors is provided;
10. A system hierarchy is provided, identifying all modules in the system;
11. All modules and duplicate functions are identified;
12. All modules called by more than one other module are identified;
13. The constants used in the system are defined only once.

For each module, system developers must check to insure that the following guidelines are contained within the system/subsystem/program specifications and test plan

documentation:

1. Inputs, outputs, and functions being performed can be clearly identified;
2. A minimal number of data references are defined, computed or obtained from external sources;
3. All conditions and processing are defined for each decision point;
4. The type of problem report is being identified within a profile of problem reports;
5. When an error condition is detected, a message is passed to the calling module;
6. Numerical techniques used in algorithms are analyzed with regards to accuracy requirements;
7. values of input ranges are tested;
8. Conflicting requests and illegal combinations are identified and checked;
9. There is a check to see if all necessary data is available before processing begins;
10. All input is checked, reporting all errors before processing begins;
11. Loop and multiple transfer index parameters range are tested before use;
12. Subscripts range are tested before use;
13. Outputs are checked for reasonableness before processing continues;
14. The number of decision/subdecision points is known;
15. The number of conditional and unconditional branches is identified;
16. The module is not dependent on information of prior processing;
17. Any limitations of the processing that are performed by the module are identified;
18. The number of entrances into modules and number of exits from modules are known;

19. The number of references to system library routines, utilities or other system provided facilities is known;
20. The number of I/O actions and the number of calling sequence parameters, which are control variable, is known;
21. Input is passed as calling sequence parameters;
22. Output is passed back to calling module;
23. Control is returned to calling module;
24. Temporary storage is not shared with other modules;
25. A module does not mix input, output, and processing functions in the same module;
26. The number of machine dependent functions that are performed is minimized;
27. Processing data volume/value is limited;
28. A common, standard subset of a programming language is used;

If developers insured that the system exhibited these traits (specified in the checklists), then they would be better assured of achieving a more reliable and maintainable system. The organizing system would update the AMT database, derive the metric scores, and generate the reports. This would serve as a more quantitative basis on which to make a decision about the progress of the system development.

At this point it became infeasible to try to collect module level information because the AMT is not yet available for AFLC. However, if a checklist is provided to programmers, then it will encourage goal-directed programming. Therefore, the software is more apt to

exhibit those desired quality traits (Ref 27). Before coding, programmers should be given a checklist specifying the following conditions for each module which contributes to reliable and maintainable software:

1. Machine level language statements should be minimized;
2. Unconditional branching should be minimized;
3. A specified structured language will be used;
4. GOTO's should be avoided;
5. Prologue comments should be provided which contain information about the function, author, version number, date, inputs, outputs, assumptions, and limitations of the modules;
6. There should be a comment which indicates which itemized requirement is satisfied by the module;
7. All decision points and transfers of control should be commented;
8. Machine language code must be commented;
9. All non-standard HOL statements must be commented;
10. All declared variables must be described by comments
11. All variable names (mnemonics) must be descriptive of the physical or functional property they represent;
12. Code should be logically blocked and indented;
13. No line should contain more than one statement;
14. Inputs should be range tested;
15. Possible conflicts or illegal combinations in inputs should be checked;
16. Parameters which are passed to or from other modules must be defined in the module;

17. Global variables must be used consistently with respect to units or type;
18. Each variable should be used for one purpose;
19. Loop and multiple transfer index parameters must be range tested before use;
20. Subscript values are range tested before use;
21. When an error condition occurs, information must be passed to the calling module;
22. Results of a computation must be checked before outputting or processing continues;
23. During execution, outputs must be within accuracy tolerances.

By including these checklists as a supplement to development procedures, management can do much to improve the efficiency and effectiveness of the SQA program that uses the AMT because these checklists correspond to the requirements for the metric worksheets. By obtaining item information from the worksheets, which are descriptive in nature, and developing the checklists to be used before each SDLC phase, SQM becomes more prescriptive, thus aiding even more to the delivery of quality software.

Chapter VII summarizes this research effort and makes recommendations to AFLC/LM concerning its SQA program.

CHAPTER VII

THESIS RECOMMENDATIONS: GUIDELINE FOR A SQA PROGRAM

The overall objective of this thesis has been to provide guidelines to AFLC/LM concerning the way Software Quality Metrics can be integrated into its SQA program.

Research Summary

Chapter I provided the background information which identified the difficulties AFLC has experienced during its attempts to establish a SQA program. It pinpointed specific symptoms that indicate AFLC lacks a viable SQA function.

In Chapter II, Software Quality Metrics were presented as a means to provide a feasible solution to this problem. The measurement concepts of SQM were shown to complement other software tools and techniques, and to identify quality characteristics that these other aids failed to quantify. The chapter focused on how SQM can be used to identify software quality requirements.

Chapter III presented the methods used in this thesis to derive the information needed to make recommendations concerning the development of AFLC/LM's SQA program. It outlined the research methodology required to demonstrate

the feasibility of incorporating Software Quality Metrics into AFLC's SQA program.

Chapter IV accomplished the first goal of this research effort: to determine which measurement tools and techniques should be used by AFLC/LM in establishing its SQA program. It concluded that the minimum set of tools are: the Automated Measurement Tool (AMT), Standards, Standards Analyzer, Dynamic Analyzer, Language Processor and a Test Bed, and that the minimum set of techniques should be: Software Quality Metrics (SQM), Reviewing, Auditing, Design Inspections, Walk-Throughs and Standardization. By including SQM and the AMT in this minimum set of tools and techniques the SQA group will be able to quantitatively assess software quality. Chapter IV also justified the employment of a toolsmith within the SQA organization.

Chapter V proposed a systemic model of the software system development effort that can be used to conceptualize the specific information requirements of the subsystems within the phases of the SDLC and thereby accomplished the second goal of this thesis: to develop a model of the decision-making process of the software system development effort that incorporates the application of SQM and which illustrates the feasibility of implementing a quantitatively oriented SQA program at AFLC/LM. As a result of this modeling effort, software development has been envisioned as a controlled management process in which control is exercised through reviews, status reporting, and

software products delivered during the SDLC. Currently, the major emphasis of the control function is to evaluate the schedule and cost performance and to determine the functional correctness of the software being developed. The concept underlying software quality metrics is to use these control vehicles to provide an indication (and therefore a mechanism of control) of the quality of the software product to be delivered (Ref 17).

Chapter VI focused on accomplishing the third goal: to demonstrate how the SQM concepts can be applied at AFLC/LM. It achieved this by providing checklists which can be used prior to each phase of the SDLC. The intent of the checklists is to emphasize the fact that the metrics, which establish the quantitative assessment of software quality, are based on the availability of key information which exhibits the characteristics of the software.

It has been found that the costs throughout the total life cycle are more affected by the characteristics of the software system than by the mission-oriented functions performed by the software system (Ref 35). Large software systems have sometimes proven untestable, unmodifiable, and largely unusable by operations personnel because of the characteristics of the software (Ref 36).

Research Recommendations

It is a function of the software quality assurance program to insure that the characteristics of the software system satisfy the requirements of the end-user. However,

by solely relying upon its current SQA tools and techniques, AFLC/LM cannot provide an indication of the quality of the software it delivers. Therefore, based on the findings of this research effort, AFLC/LM should incorporate the concepts of Software Quality Metrics (SQM) into its software development effort. To accomplish this AFLC should:

1. Acquire the Automated Measurement Tool (AMT) which is needed to operationalize the SQM methodology. To obtain the tool, a request to Joe Cavano (Autovon 587-7834) at the Rome Air Development Center (RADC) should be made to acquire the software and tapes for the AMT. Also information concerning implementation, testing and validation should be acquired.

2. Develop checklists, similar to the ones in Chapter VI, to be used by software developers to insure that they are aware of the software traits which affect all factors of software quality. These checklists should be included as a supplement to the Project Manager's Handbook, which is now being developed (Ref 89). As a minimum, the checklists should include coverage of all questions in the metric worksheets in Appendix A.

3. Contract with General Electric's Command and Information Systems Division at Sunnyvale, CA:

- a) to train SQA personnel in procedures to fully utilize the SQM methodology and the Automated Measurement Tool (AMT);

b) to establish standards and procedures that can be used by the SQA program in assessing the quality of software products, and

c) to interface existing software tools at AFLC to the AMT.

4. Assign an individual in the SQA program to be responsible for the development and maintenance of written procedures and a tool library. This toolsmith should be familiar with other software tools and be able to determine (through procedures in Chapter IV) if acquisition of the tool is justified.

The nature of the framework for Software Quality Metrics allows an upper level application of the SQM concepts without the acquisition of the AMT. The structured (step-wise) procedures, discussed in Chapter II and depicted in Figure 9, provide the ability to specify system requirements in software-oriented terms. This provides goal-directed design and implementation which has been shown to improve the quality of the end product. This fact alone means AFLC/LM now has the capability to significantly impact the quality of its software systems before development begins.

Once it acquires the AMT, AFLC can expand its capability to monitor and control the software development effort. The AMT will serve as the core for a Decision Support System to provide management with an objective

assessment of the system development. By integrating other software tools with the AMT, AFLC can effectively automate the monitoring function of the SDLC, thus reducing the manpower requirements for its SQA program.

In conclusion, by acquiring the Automated Measurement Tool to support the application of Software Quality Metrics, AFLC/LM will significantly improve the quality of the software systems it develops because it will be able to quantitatively measure the quality of the software system as it is being developed. There can be no justification for not acquiring this assessment capability.

REFERENCES

REFERENCES USED

1. Proceedings of the 1973 Symposium on the High Cost of Software, September 1973.
2. "Findings and Recommendations of the Joint Logistics Commander," Software Reliability Working Group, November 1975.
3. "Government/Industry Software Sizing and Costing Workshop," Summary Notes, USAFESD, October 1974.
4. Myers, G.J. "Characteristics of Composite Design," DATAMATION, September 1973.
5. Myers, W. "The Need for Software Engineering," Computer, February 1973.
6. Cooper, J. and M. Fisher, ed. Software Quality Management, New York: Petrocelli Books, Inc., 1979.
7. Reifer, D.J. "Tutorial: Software Management," IEEE Catalog No. EHO 146-1, 1979.
8. Freeman, P. & A.I. Wasserman. "Tutorial on Software Design Techniques," IEEE Catalog No EHO 161-0, 1980.
9. Cho, C. An Introduction to Software Quality Control, New York: John Wiley and Sons, 1980.
10. Biggs, C.L., E. Birks, and W. Atkins. Managing the Systems Development Process, Englewood Cliffs, N.J.: Prentice Hall, Inc., 1980.
11. Fohrman, W.G. "Observations on Configuration Management," Software Configuration Management, IEEE Catalog No EHO 169-3, 1980.
12. Jensen, R.H. & C.C. Jonies. Software Engineering, Englewood Cliffs, N.J.: Prentice Hall, Inc., 1979.
13. Ruby, R.J., J.A. Dana, and P. Biche. "Quantitative Aspects of Software Validation," IEEE Transactions on Software Engineering, June 1975.
14. Ruby, R.J. "Software/System Acquisition," Graduate Lecture Notes at AFIT for EE 5.45, September, 1981.

15. Reifer, D.J. "Software Quality Assurance Tools and Techniques," Software Quality Management, New York: Petrocelli Books, Inc., 1979.
16. McCall, J.A. "An Assessment of Current Software Metric Research," EASCON-80 Presentation, September, 1980.
17. Richards, P.K., G. Walters, & J.A. McCall. "Factors in Software Quality," Three volumes, NTIS, AD-A049-014, AD-A049-015, AD-A049-055., November 1977.
18. Cavano, J. & J. McCall. "A Framework for the Measurement of Software Quality," Proceedings, Software Quality Assurance Workshop, November 1978.
19. Boehm, B., et al. "Quantitative Evaluation of Software Quality," Proceedings, 2nd International Conference of Software Engineering, October 1976.
20. Rubey, R. & R. Hartwick. "Quantitative Measurement of Program Quality," Proceedings, 23rd National Conference, ACM, 1968.
21. Kosarajo, S. and H. Ledgard. "Concepts in Quality Design," NBS Technical Note 842, August 1974.
22. Boehm, B., et al. Characteristics of Software Quality: New York: North Holland Publishing Co., 1978.
23. McCall, J.A. and M.T. Matsumoto. Software Quality Metrics Enhancements, Vol. I, Contract # F30602-78-C-0216, September 1979.
24. Walters, G.F. & J.A. McCall. "The Development of Metrics for Software Reliability and Maintainability," Proceedings, Annual Reliability and Maintainability Symposium, January 1978.
25. Reifer, D. "Toward Specifying Software Properties," IFIP Conference on Modeling of Environmental Systems, Tokyo, Japan, April 1976.
26. Gilb, T. Software Metrics, Cambridge, Mass: Winthrop Publishers, Inc., 1976.
27. Weinberg, G. "The Psychology of Improved Programming Performance," DATAMATION, November 1972.
28. Matsumoto, M. & J. McCall. "Software Quality Measurement Manual," RADC-TR-80-109, Vol. II, April 1980.
29. Fagan, M.E. "Design and Code Inspections to Reduce Errors in Program Development," IBM Systems Journal, Vol. 16, No. 3, 1976.

30. Myer, G. "A controlled experiment in program testing and code walk-throughs/inspections," Communications of the ACM, Vol. 21, September 1978.
31. Walters, G.F. "Application of Metrics to Predict Software Quality," MIDCON SQA Program, 1979.
32. Boehm, B.W. "Software and Its Impact: A Quantitative Approach," Datamation, April 1973.
33. Elshoff, J.L. "Measuring Commercial PL/I Programs Using Halstead's Criteria," SIGPLAN Notices, May 1976.
34. Kosarajo, S.R. and H.F. Ledgard. "Concepts in Quality Software Design," NBS Technical Note 842, August 1974.
35. Fitzsimmons, A. and T. Love. "A Survey of Software Practitioners to Identify Critical Factors in the Software Development Process," GE TIS 76ISPO03, December 1976.
36. Millstein, R.E., et al. "On Program Transferability," Applied Data Research, Inc., RADC-TR-70-217, November 1970.
37. "Quality Assurance," Department of Defense Directive 4155.1 (1972), Enclosure 2.
38. Woodward, Richard. Interviews from April 1981 to February 1982 in AFLC/LM, Wright-Patterson AFB, OH.
39. Fisher, M.J., et al. "Software Quality Assurance and Reliability as it Relates to Configuration Management," Report of the Eleventh Annual EIA Data and Configuration Management Workshop, Panel No. 7, San Diego, CA, October 17-21, 1977.
40. Ruby, Raymond J. "Software Quality Assurance and Verification and Validation," Seminar Document from the DPMA symposium, April 23-24, 1981.
41. McCabe, Thomas J. and Frederick Stern. "Use of Metrics to Measure Quality," Conference Proceedings from the DPMA National Symposium on Effective Methods of EDP Quality Assurance, Chicago, IL., April 1-3, 1981.
42. Stratton, Cloyd D. Interviews from September to November, 1981 in AFLC/LM, Wright-Patterson AFB, OH.
43. Rullo, Thomas A. Advances in Computer Programming Management, Vol. I, Philadelphia: Heyden & Son, Inc., 1980.

44. Ruby, Raymond L. "Software Quality Assurance," Technical Session 1A, Air Force Institute of Technology (AFIT) Association of Graduates (AOG) Biennial Symposium, AFIT/Department of Electrical Engineering, Wright-Patterson AFB, OH, November 18, 1981.
45. Kerzner, Harold. Project Management: A Systems Approach to Planning, Scheduling, and Controlling, New York: Van Nostrand Reinhold Company, 1979.
46. DoD Standard 7935.1-S, Automated Data Systems Documentation Standards, Government Printing Office, Washington, D.C., September 13, 1977.
47. AFR 300-15, Automated Data System Project Management, U.S. Government Printing Office, January 16, 1978.
48. Tinsley, Jack. Interviews at AFLC/LM about Configuration Management, Program Management, and Application of Standards, August - December 1981.
49. Schoderbek, C.G., et al. Management Systems: Conceptual Considerations, Dallas: Business Publications, Inc., 1980.
50. Bryan, William, et al. "Tutorial: Software Configuration Management," IEEE Catalog No. EHO 169-3, 1980.
51. Markham, Dave and James McCall. Two Days of Interviews on General Electric's Software Quality Metrics, October 29-30, 1981.
52. Whited, Jon A. "Management Control Practices for Software Quality," Software Quality Management, New York: Petrocelli Books, Inc., 1979.
53. Thayer, L. Communication and Communication Systems, Homewood, IL: Richard D. Irwin, Inc., 1978.
54. Simon, H.A. The Sciences of the Artificial, Cambridge, Mass.: The MIT Press, 1969.
55. Wood, Dennis L. "Department of Defense Software Quality Requirements," Software Quality Management, New York: Petrocelli Books, Inc., 1979.
56. Gustafson & Kerr. "Some Practical Experience with a Software Quality Assurance Program," Communications of the ACM, January 1982.

57. Simon, H.A. The Shape of Automation for Men and Management, New York: Harper Torchbooks, The Academy Library, 1965.
58. Clark, Thomas. "Logistics Decision Support Systems," LM 6.15, Lecture Notes for course in Air Force Institute of Technology, School of Systems and Logistics, 1981.
59. Casey, J.K. "The Changing Role of the In-House Computer Application Software Shop," GE TIS #74AEG195, February 1974.
60. Dennis, J.B., G. Goos, J. Poole, C.C. Gotlieb, et al. "Advanced Course on Software Engineering," Springer-Verlag, New York, 1970.
61. Lieblein, E. "Computer Software: Problems and Possible Solutions," CENTACS USAECOM Memorandum, November 7, 1972.
62. Kernighan, B. and P. Plauger. The Elements of Programming Style, McGraw-Hill, 1974.
63. Culpepper, L.M. "A System for Reliable Engineering Software," International Conference on Reliable Software, 1975.
64. Hague, S.J. and B. Ford. "Portability-Prediction and Correction," Software Practices & Experience, Vol. 6, 61-69, 1976.
65. Kosy, D. "Air Force Command and Control Information Processing in the 1980s: Trends in Software Technology," Rand, June 1974.
66. Mealy, G.H., D.J. Farber, E.E. Morehoff, and Sattley. "Program Transferability Study," RADC, November 1968.
67. Edwards, N.P.. "The Effect of Certain Modular Design Principles on Testability," International Conference on Reliable Software, 1975.
68. Liskov, B.H. "Guidelines for the Design and Implementation of Reliable Software Systems," MITRE Report 2345, February 1973.
69. Light, W. "Software Reliability/Quality Assurance Practices," Briefing given at AIAA Software Management Conferences, 1976.
70. Pathway Program. Product Quality Assurance for Shipboard Installed Computer Programs, Naval Sea Systems Command, April 1976.

71. Goodenough, J. "Exception Handling Design Issues," SIGPLAN Notices, July 1975.
72. Richards, P., et al. "Simulation Data Processing Study: Language and Operating System Selection," GE TIS 74CIS09, June 1974.
73. Marshall, S., R.E. Millstein, and K. Sattley. "On Program Transferability," Applied Data Research, Inc., RADC-TR-70-217, November 1970.
74. "Support of Air Force Automatic Data Processing Requirements through the 1980's," SADPR-85, July 1973.
75. Myers, G.J. Reliable Software through Composite Design, Petrocelli/Charter, 1975.
76. Myers, G.J. Software Reliability: Principles and Practices, John Wiley & Sons, New York, 1976.
77. "SAMSO Program Management Plan Computer Program Test and Evaluation," February 1975.
78. Thayer, T.A., W.L. Hetrick, M. Lipow, and G.R. Craig. "Software Reliability Study," RADC TR-76-238, August 1976.
79. "U.S. Army Integrated Software Research and Development Program," USACSC, January 1975.
80. Schonfelder, J.L. "The Production of Special Function Routines for a Multi-Machine Library," Software-Practice and Experience, Vol. 6, 1976.
81. Whipple, L. "AFAL Operational Software Concept Development Program," Briefing given at Software Subpanel, Joint Deputies for Laboratories Committee, February 12, 1975.
82. Wulf, W.A. "Report of Workshop 3 - Programming Methodology," Proceedings of a Symposium on the High Cost of Software, September 1973.
83. Yourdon, E. Techniques of Program Structure and Design, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.
84. McCall J. and Markham. "Automation of Quality Metrics: Final Report," Contract # F30602-79-C-0267, December, 1981.
85. AFR 300-12, "Procedures for Managing Automatic Data Processing Systems (ADPS), U.S. Government Printing Office, September, 1977.

86. MIL-S-52779A, "Software Quality Assurance Program Requirements," U.S. Government Printing Office, September, 1979.
87. TRW, Airborn Systems Software Acquisition Engineering Guidebook for Quality Assurance, Report No. 30323-6005-TU-00, November 1977.
88. Markham, D., et al. "The Automated Measurement of Software Quality," Paper submitted to COMSAC 81, June 1981.
89. "Project Manager's Handbook," Working Copy, for AFLC's Software Development, Undated.
90. Reifer, Donald J. and Stephen Trattner. "A Glossary of Software Tools and Techniques," Computer, July 1977.
91. Reynolds, C. and R.T. Yeh. "Induction as the Basis for Program Verification," IEEE Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976.
92. Good, D.I., R.L. London, and W.W. Bledsoe. "An Inter-Active Verification System," Proceedings of the International Conference on Reliable Software, IEEE, April 1975.
93. Tsui, Frank and Lew Priven. "Implementation of Quality Control in Software Development," Proceedings of the 1976 National Computer Conference, AFIPS Press, 1976.
94. Miller, Edward F., Jr. "Program Testing: Art Meets Theory," Computer, July 1977.
95. Howden, William E. "Methodology for the Generation of Program Test Data," IEEE Transactions on Software Engineering, Vol. SE-3, 1977.
96. Fairley, Richard E. "Tutorial: Static Analysis and Dynamic Testing of Computer Software," Computer, April 1978.
97. Reifer, Donald and Robert L. Ettenger. Test Tools: Are They a Cure-all? SAMSO-TR-75-13, October 15, 1974.
98. Freeman, Peter. "Toward Improved Review of Software Designs," Proceedings of the 1975 National Computer Conference, AFIPS Press, 1975.
99. Bratman, Harvey and Marcia C. Finfer. Software Acquisition Management Guidebook: Verification, ESD-TR-77-263, August 1977.

100. Reifer, Donald J. A Structured Approach to Modeling Computer Systems, SAMSO-TR-75-3, August 30, 1974.
101. MIL-STD-483 (USAF), Configuration Management Practices for Systems, Equipment, Munitions and Computer Programs, June 1971.
102. National Bureau of Standards, Guidelines for Documentation of Computer Programs and Automated Data Systems, Federal Information Processing Standards Publication 38, February 1976.
103. MIL-STD-1589 (USAF), JOVIAL (J73/I), February 28, 1977.
104. "Draft Proposed ANS FORTRAN, BSR X3.9, X3J3/76," SIGPLAN Notices, Vol. II, No. 3, March 1976.
105. Stevens, W.P., G.J. Meyers, and L.L. Constantine. "Structured Design," IBM Systems Journal, No. 2, 1974.
106. Peters, L.J. and L.L. Trip. "Comparing Software Design Methodologies," Datamation, November 1977.
107. Structured Programming Series, USAF Rome Air Development Center, Vols. 1-15, July 1975.
DDC Accession Numbers Follows:
"Programming Language Standards," AD-A016 771.
"Pre-Compiler Specifications," AD-A018 046.
"Pre-Compiler Program Documentation," AD-A013 255.
"Data Structuring," AD-A015 794.
"Program Support Library Requirements," AD-A003 339.
"Program Support Library Program Specifications," AD-A007 796.
"Documentation Standards," AD-A016 414.
"Program Design Study," AD-A016 415.
"Management Data Collection and Reporting," AD-A008 640.
"Chief Programmer Team Operations," AD-A008 861.
"Estimating Software Resource Requirements," AD-A016 416.
"Training Materials," AD-A026 947.
"Software Tool Impact," AD-A015 795.
"Validation and Verification," AD-A016 668.
"Final Report," AD-A020 858.
108. Darringer, John A. and James C. King. "Applications of Symbolic Execution to Program Testing," Computer, April 1978.
109. Howden, W.E. "Symbolic Testing and the DISSECT Symbolic Evaluation System," IEEE Transaction on Software Engineering, Vol. SE-3, 1977.

110. Waldstein, N.S. The Walk-Thru-A Method of Specification, Design and Review, IBM Corporation Technical Report TR 00.2536, June 1974.
111. Logicon Technical Staff, Management Guide to Avionics Software Acquisition: Volume IV -- Technical Aspects Relative to Software Acquisition, ASD-TR-76.11, Vol. IV, June 1976.
112. Hoffman, R.H. "NASA/Johnson Space Center Approach to Automated Test Data Generation," Proceedings of Computer Science and Statistics: Eighth Annual Symposium on the Interface, available from UCLA, February 1975.
113. Panzl, David J. "Automatic Software Test Drivers," Computer, April 1978.
114. Trattner, S. Tools for Analysis of Software Security, Aerospace Corporation, ATR-77 2740-1, October 15, 1976.
115. Landes, Michael. "Consistency Checker," Summary, Presentation at the International DOD/Industry Conference on Software Verification and Validation, Rome Air Development Center, August 1976.
116. Callender, E.D., M. Feliciano, and L.D. Jennings. SAMSO Computer Language and Software Development Environment Requirements, SAMSO-TR-290, 1975.
117. Felty, James L. and Martin S. Roth. Software Support Tools, Intermetrics Inc., IR-204-2, October 15, 1976.
118. Couger, J. Daniel. "Evaluation of Business System Analysis Techniques," Computing Surveys, Vol. 5, No. 3, September 1973.
119. DeWolf, J. Barton and Jonathan Wexler. "Approaches to Software Verification with Emphasis on Real-Time Applications," Proceedings of the AIAA/NASA/IEEE/ACM Computers in Aerospace Conference, October 31 - November 2, 1977.
120. Whipple, L.K. and M.A. Pitts. User's Appraisal of an Automated Program Verification Aid, AFAL-TR-75-242, December 1975.
121. Spanbauer, Robert N. (USAF). "The F16 Software Development Program," Proceedings of Conference on Managing the Development of Weapon System Software, Maxwell Air Force Base, Alabama, May 12-13, 1976.
122. Nassi, I. and B. Schneiderman. "Flowchart Techniques

for Structured Programming," SIGPLAN Notices,
August 1973.

123. Reifer, Donald J. Interim Report on the Aids Inventory Project, SAMSO-TR-74-184, July 16, 1975.
124. Nutt, Gary J. "Tutorial: Computer System Monitors," Computer, November 1975.
125. Highland, Harold J. (ed.) Computer Performance Evaluation, U.S. Department of Commerce, National Bureau of Standards, Special Publication 401, September 1974.
126. Baum, J.D. and J.B. DiStefano. "Avionics In-Flight System/Software Test Tool -- Anomaly Trace," Proceedings of the Aeronautical Systems Software Workshop, Dayton, Ohio, April 2-4, 1974.
127. Hamilton, M. and S. Zeldin. "Higher Order Software -- A Methodology for Defining Software," IEEE Transactions on Software Engineering, Vol. SE-2, No. 1, March 1976.
128. Bell, Thomas E., David C. Bixler, and Margaret E. Dyer. "An Extendable Approach to Computer-Aided Software Requirements Engineering," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977.
129. Caine, Stephen H. and E. Kent Gordon. "PDL -- A Tool for Software Design," Proceedings of the 1975 National Computer Conference, AFIPS Press, 1975.
130. Baker, F. Terry. "Structured Programming in a Production Programming Environment," IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, June 1975.
131. Poseidon MK88 Fire Control System Computer Program Verification and Validation Techniques Study, Volume III, Ultrasystems Inc., Newport Beach, CA, November 1973.
132. Malcolm, Donald G. "Cost-Effective Management Information Systems," Management Information Systems Short Course Notes, Engineering 819.39, University of California at Los Angeles, May 1978.
133. Hardy, I. Trotter, Belkis Leong-Hong, and Dennis W. Fife. Software Tools: A Building Block Approach, U.S. Department of Commerce, National Bureau of Standards, Special Publication 500-14, August 1977.

134. Teichroew, D. and E.A. Hershey, III. "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977.
135. Basili, Victor R. "A Panel Session - User Experience with the New Software Methods," Proceedings of the 1978 National Computer Conference, AFIPS Press, 1978.
136. Ramamoorthy, C.V. and K.H. Kim. "Software Monitors Aiding Systematic Testing and Their Optional Placement," Proceedings of the 1st National Conference on Software Engineering, IEEE Catalog No. 75CH0992-8C, September 1975.
137. Herring, F.P. and C.J. Mabee. Survey of Support Software for Operational Flight Programs and Avionics Integration Support Facility Software, TRW Systems, Report 28675-6232-RU-00, May 1977.
138. Chambers, T.V. "Shuttle Avionics Integration Laboratory," Proceedings of the AIAA/NASA/IEEE/ACM Computers in Aerospace Conference, October 31 - November 2, 1977.
139. Hollowich, Michael and Frank Borasz. "The Software Design & Verification System (SDVS), An Integrated Set of Software Development and Management Tools," NAECON '76 Record, IEEE Catalog No. 76CH1082-7 NAECON, 1976.
140. Jelinski, P.B., et al. "Metrics of Software Quality," DTIC Technical Report # ADA093788, November 1980.
141. "Configuration Management of Software Programs," A Continuing Engineering Education Program at George Washington University, November 1978.
142. Brooks, Robert. Interviews at AFLC/LMX about Configuration Management, Project Coordination, and G072A MMSIP, October - December 1981.

APPENDIX A

METRIC WORKSHEETS

(Refs 17,23)

METRIC WORKSHEET 1 REQUIREMENTS ANALYSIS/SYSTEM LEVEL		SYSTEM NAME		DATE	
				INSPECTOR	
I. COMPLETENESS (CORRECTNESS, RELIABILITY)					
1. Number of major functions identified (equivalent to CPCI). CP.1					
2. Are requirements itemized so that the various functions to be performed, their inputs and outputs, are clearly delineated? CP.1(1)				Y	N
3. Number of major data references. CP.1(2)					
4. How many of these data references are not defined? CP.1(2)					
5. How many defined functions are not used? CP.1(3)					
6. How many referenced functions are not defined? CP.1(4)					
7. How many data references are not used? CP.1(2)					
8. How many referenced data references are not defined? CP.1(6)					
9. Is the flow of processing and all decision points in that flow described? CP.1(5)				Y	N
10. How many problem reports related to the requirements have been recorded? CP.1(7)					
11. How many of those problem reports have been closed (resolved)? CP.1(7)					
II. PRECISION (RELIABILITY)					
1. Has an error analysis been performed and budgeted to functions? AY.1(1)				Y	N
2. Are there definitive statements of the accuracy requirements for inputs, outputs, processing, and constants? AY.1(2)				Y	N
3. Are there definitive statements of the error tolerance of input data? ET.2(1)				Y	N
4. Are there definitive statements of the requirements for recovery from computational failures? ET.3(1)				Y	N
5. Is there a definitive statement of the requirement for recovery from hardware faults? ET.4(1)				Y	N
6. Is there a definitive statement of the requirements for recovery from device errors? ET.5(1)				Y	N
III. SECURITY (INTEGRITY)					
1. Is there a definitive statement of the requirements for user input/output access controls? AC.1(1)				Y	N
2. Is there a definitive statement of the requirements for data base access controls? AC.1(2)				Y	N
3. Is there a definitive statement of the requirements for memory protection across tasks? AC.1(3)				Y	N
4. Is there a definitive statement of the requirements for recording and reporting access to system? AA.1(1)				Y	N
5. Is there a definitive statement of the requirements for immediate indication of access violation? AA.1(1)				Y	N

METRIC WORKSHEET 1 REQUIREMENTS ANALYSIS/SYSTEM LEVEL	SYSTEM NAME _____	DATE _____ INSPECTOR: _____
IV. HUMAN INTERFACE (USABILITY)		
1. Are all steps in the operation described (operations concept)? OP.1(1)	Y	N
2. Are all error conditions to be reported to operator/user identified and the responses described? OP.1(2)	Y	N
3. Is there a statement of the requirement for the capability to interrupt operation, obtain status, modify, and continue processing? OP.1(3)	Y	N
4. Is there a definitive statement of requirements for optional input media? CM.1(6)	Y	N
5. Is there a definitive statement of requirements for optional output media? CM.2(7)	Y	N
6. Is there a definitive statement of requirements for selective output control? CM.2(1)	Y	N
V. PERFORMANCE (EFFICIENCY)		
1. Have performance requirements (storage and run time) been identified for the functions to be performed? EE.1	Y	N
VI. SYSTEM INTERFACES (INTEROPERABILITY)		
1. Is there a definitive statement of the requirements for communication with other systems? CC.1(1)	Y	N
2. Is there a definitive statement of the requirements for standard data representations for communication with other systems? DC.1(1)	Y	N
VII. INSPECTOR'S COMMENTS		
Make any general or specific comments that relate to the quality observed while applying this checklist.		

DESIGN WORKSHEET FOR DESIGN/SYSTEM LEVEL	SYSTEM NAME _____	DATE: _____ INSPECTOR: _____																																												
IV. OPTIMIZATION (EFFICIENCY)																																														
1. Are storage requirements allocated to design? SE.1(1) 2. Are virtual storage facilities used? SE.1(2) 3. Is dynamic memory management used? SE.1(5) 4. Is a performance optimizing compiler used? SE.1(7) 5. Is global data defined once? CS.2(3) 6. Have Data Base or files been organized for efficient processing? EE.3(5) 7. Is data packing used? EE.2(5) 8. Number of overlays EE.2(4) 9. Overlay efficiency - memory allocation EE.2(4) max overlay size min overlay size	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 50%;">Y</td><td style="width: 50%;">N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </table>	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N									<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 50%;">Y</td><td style="width: 50%;">N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </table>	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N								
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
V. SECURITY (INTEGRITY)																																														
1. Are user Input/Output access controls provided? AC.1(1) 2. Are Data Base access controls provided? AC.1(2) 3. Is memory protection across tasks provided? AC.1(3) 4. Are there provisions for recording and reporting errors? AC.2(1,2)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 50%;">Y</td><td style="width: 50%;">N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> </table>	Y	N	Y	N	Y	N	Y	N	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 50%;">Y</td><td style="width: 50%;">N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> </table>	Y	N	Y	N	Y	N	Y	N																												
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
VI. SYSTEM INTERFACES (INTEROPERABILITY)																																														
1. How many other systems will this system interface with? CC.1(1) 2. Have protocol standards been established? CC.1(2) 3. Are they being complied with? CC.1(2) 4. Number of modules used for input and output to other systems? CC.1(3,4) 5. Has a standard data representation been established or translation standards between representations been established? DC.1(1) 6. Are they being complied with? DC.1(2) 7. Number of modules used to perform translations? DC.1(3)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 50%;">Y</td><td style="width: 50%;">N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </table>	Y	N	Y	N	Y	N	Y	N	Y	N							<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 50%;">Y</td><td style="width: 50%;">N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td>Y</td><td>N</td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </table>	Y	N	Y	N	Y	N	Y	N	Y	N																		
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
Y	N																																													
VII. HUMAN INTERFACE (USABILITY)																																														
1. Are all steps in operation described including alternative flows? OP.1(1) 2. Number of operator actions? OP.1(4)	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 50%;">Y</td><td style="width: 50%;">N</td></tr> <tr><td> </td><td> </td></tr> </table>	Y	N			<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="width: 50%;">Y</td><td style="width: 50%;">N</td></tr> <tr><td> </td><td> </td></tr> </table>	Y	N																																						
Y	N																																													
Y	N																																													

METRIC WORKSHEET 2a DESIGN/SYSTEM LEVEL		SYSTEM NAME: _____		DATE INSPECTOR: _____	
VIII. TESTING (TESTABILITY) - APPLY TO TEST PLAN, PROCEDURES, RESULTS (CONTINUED)					
6. Number of interfaces to be tested? IN.2(1)		9. Number of modules? IN.3(1)			
7. Number of itemized performance requirements? IN.2(2)		10. Number of modules to be exercised? IN.3(1)			
8. Number of performance requirements to be tested? IN.2(2)		11. Are test inputs and outputs provided in summary form? IN.3(2)			
IX. DATA BASE					
1. Number of unique data items in data base SI.1(6)					
2. Number of preset data items SI.1(6)					
3. Number of major segments (files) in data base SI.1(7)					
X. INSPECTOR'S COMMENTS					
Make any general or specific comments about the quality observed while applying this checklist.					

METRIC WORKSHEET 2a DESIGN/SYSTEM LEVEL	SYSTEM NAME: _____	DATE INSPECTOR: _____
VIII. TESTING (TESTABILITY) - APPLY TO TEST PLAN, PROCEDURES, RESULTS (CONTINUED)		
6. Number of interfaces to be tested? IN.2(1)		9. Number of modules? IN.3(1)
7. Number of itemized performance requirements? IN.2(2)		10. Number of modules to be exercised? IN.3(1)
8. Number of performance requirements to be tested? IN.2(2)		11. Are test inputs and outputs provided in summary form? IN.3(2)
IX DATA BASE		
1. Number of unique data items in data base SI.1(6)		
2. Number of preset data items SI.1(6)		
3. Number of major segments (files) in data base SI.1(7)		
X INSPECTOR'S COMMENTS		
Make any general or specific comments about the quality observed while applying this checklist.		

METRIC WORKSHEET 2b DESIGN/MODULE LEVEL	SYSTEM NAME: _____ MODULE NAME: _____	DATE: _____ INSPECTOR: _____

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

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

III. STRUCTURE (RELIABILITY, MAINTAINABILITY, TESTABILITY)		
1. How many Decision Points are there? SI.3		1. How many conditional branches are there? SI.3
2. How many subdecision Points are there? SI.3		1. How many unconditional branches are there? SI.3

METRIC WORKSHEET 2B DESIGN/MODULE LEVEL	SYSTEM NAME: _____ MODULE NAME: _____	Pg. 2					
III. STRUCTURE (RELIABILITY, MAINTAINABILITY, TESTABILITY) (CONTINUED)							
5. Is the module dependent on the source of the input or the destination of the output? SI.1(3)	<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N	7. Are any limitations of the processing performed by the module identified? EX.2(1)	<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N
Y	N						
Y	N						
6. Is the module dependent on knowledge of prior processing? SI.1(3)	<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N	8. Number of entrances into modules SI.1(5)			
Y	N						
		9. Number of exits from module SI.1(5)					
IV. REFERENCES (MAINTAINABILITY, FLEXIBILITY, TESTABILITY, PORTABILITY, REUSABILITY, INTEROPERABILITY)							
1. Number of references to system library routines, utilities or other system provided facilities SS.1(1)		8. Is temporary storage shared with other modules? MD.2(7)					
2. Number of input/output actions MI.1(2)		9. Does the module mix input, output and processing functions in same module? GE.2(1)					
3. Number of calling sequence parameters MD.2(3)		10. Number of machine dependant functions performed? GE.2(2)					
4. How many calling sequence parameters are control variables MD.2(3)		11. Is processing data volume limited? GE.2(3)	<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N		
Y	N						
5. Is input passed as calling sequence parameters MD.2(4)	<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N	12. Is processing data value limited? GE.2(4)	<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N
Y	N						
Y	N						
6. Is output passed back to calling module? MD.2(5)	<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N	13. Is a common, standard subset of programming language to be used? SS.1(2)	<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N
Y	N						
Y	N						
7. Is control returned to calling module MD.2(6)	<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N	14. Is the programming language available in other machines? MI.1(1)	<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N
Y	N						
Y	N						
V. EXPANDABILITY (FLEXIBILITY)							
1. Is logical processing independent of storage specification? EX.1(1)			<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N		
Y	N						
2. Are accuracy, convergence, or timing attributes parametric? EX.2(1)			<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N		
Y	N						
3. Is module table driven? EX.2(2)			<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N		
Y	N						
VI. OPTIMIZATION (EFFICIENCY)							
1. Are specific performance requirements (storage and routine) allocated to this module? EE.1			<table border="1" style="margin: auto;"> <tr><td>Y</td><td>N</td></tr> </table>	Y	N		
Y	N						

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

VI. OPTIMIZATION (EFFICIENCY) (CONTINUED)

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

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

3. Are non-loop dependent functions kept out of loops? EE.2(i)

4. Is bit/byte packing/unpacking performed in loops? EE.2(5)

5. Is data indexed or reference efficiently? EE.3(5)

VII. FUNCTIONAL CATEGORIZATION

Categorize function performed by this module according to following:

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

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

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

☐ **ALGORITHM** - a module whose prime function is computation.

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

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

VIII. CONSISTENCY

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

METRIC WORKSHEET 2b DESIGN/MODULE LEVEL	SYSTEM NAME: _____ MODULE NAME: _____	Pg. 4
IX. INSPECTOR'S COMMENTS		
Make any specific or general comments about the quality observed while applying this checklist?		

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

I. STRUCTURE (RELIABILITY, MAINTAINABILITY, TESTABILITY)

1. Number of lines of code MD.2(2) _____ 2. Number of lines excluding comments SI.4(2) _____ 3. Number of machine level language statements SD.3(1) _____ 4. Number of declarative statements SI.4 _____ 5. Number of data manipulation statements SI.4 _____ 6. Number of statement labels SI.4(6) _____ 7. Number of entrances into module SI.1(5) _____ 8. Number of exits from module SI.1(5) _____ 9. Maximum nesting level SI.4(7) _____ 10. Number of decision points (IF, WHILE, REPEAT, DO, CASE) SI.3 _____	11. Number of sub-decision points SI.3 _____ 12. Number of conditional branches (computed go to) SI.4(8) _____ 13. Number of unconditional branches (GOTO, ESCAPE) SI.4(9) _____ 14. Number of loops (WHILE, DO) SI.4(3) _____ 15. Number of loops with jumps out of loop SI.4(3) _____ 16. Number of loop indices that are modified SI.4(4) _____ 17. Number of constructs that perform module modification (SWITCH, ALTER) SI.4(5) _____ 18. Number of negative or complicated compound boolean expressions _____ 19. Is a structured language used SI.4(2) Y ☐ N ☐ 20. Is flow top to bottom (are there any backward branching GOTOs) SI.4(1) Y ☐ N ☐

II. CONCISENESS (MAINTAINABILITY) - SEE SUPPLEMENT

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

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

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

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

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

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

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

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

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

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

VII. ERROR HANDLING - (RELIABILITY)

1. How many loop and multiple transfer index parameters are not range tested before use? ET.3(2)		1. Number of mix mode expressions? EE.3(3)		
2. Are subscript values range tested before use? ET.3(3)		2. How many variables are initialized when declared? EE.3(2)		
3. When an error condition occurs, is it passed to the calling module? ET.1(3)	Y	3. How many loops have non-loop dependent statements in them? EE.2(1)	Y	N
4. Are the results of a computation checked before outputting or before processing continues? ET.3(4)	Y	4. How many loops have bit/byte packing/unpacking? EE.2(5) SE.1(6)	Y	N
	N	5. How many compound expressions defined more than once? EE.2(3)		

VIII. (EFFICIENCY)

1. Number of mix mode expressions? EE.3(3)		1. Number of mix mode expressions? EE.3(3)		
2. Are subscript values range tested before use? ET.3(3)		2. How many variables are initialized when declared? EE.3(2)		
3. When an error condition occurs, is it passed to the calling module? ET.1(3)	Y	3. How many loops have non-loop dependent statements in them? EE.2(1)	Y	N
4. Are the results of a computation checked before outputting or before processing continues? ET.3(4)	Y	4. How many loops have bit/byte packing/unpacking? EE.2(5) SE.1(6)	Y	N
	N	5. How many compound expressions defined more than once? EE.2(3)		

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

IX. PORTABILITY	X. FLEXIBILITY
1. Is code independent of word and character size? MI.1(3)	1. Is module table driven EX.2(2)
Y N	Y N
2. Number of lines of machine language statements. MI.1	2. Are there any limits to data values that can be processed? GE.2(4)
Y N	Y N
3. Is data representation machine independent? MI.1(4)	3. Are there any limits to amounts of data that can be processed? GE.2(3)
Y N	Y N
4. Is data access/storage system software independent? SS.1	4. Are accuracy, convergence and timing attributes parametric? EX.2(1)
Y N	Y N

XI. DYNAMIC MEASUREMENTS (EFFICIENCY, RELIABILITY)									
1. During execution are outputs within accuracy tolerances? AY.1(5)	Y N								
2. During module/development testing, what was run time? EX.2(3)	Y N								
3. Complete memory map for execution of this module SE.1(4) Size (words of memory)									
<table border="1" style="margin: auto;"> <tr><td>APPLICATION</td><td></td></tr> <tr><td>SYSTEM</td><td></td></tr> <tr><td>DATA</td><td></td></tr> <tr><td>OTHER</td><td></td></tr> </table>	APPLICATION		SYSTEM		DATA		OTHER		
APPLICATION									
SYSTEM									
DATA									
OTHER									
4. During execution how many data items were referenced but not modified EE.3(6)	Y N								
5. During execution how many data items were modified EE.3(7)	Y N								

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

APPENDIX B

WORKSHEET REQUIREMENTS

(Ref 51)

WORKSHEET REQUIREMENTS

The following is a description of the data that the Automated Measurement Tool (AMT) requires (Ref 17,23). The following conventions apply to certain documents and source code:

As an overview:

- o Worksheet 1 is manually applied to the requirements analysis at the system level.
- o Worksheet 2a is manually applied to the design document at the system level.
- o Worksheet 2b is manually applied as required to the design documents at the module level.
- o Worksheet 3 is applied to the code. Many of the values will be collected automatically by AMT and entered into the data base for further usage. The remaining items will have to be entered manually.

All data may be included in the AMT database provided the following conventions are to be used in filling out the worksheets:

1. If a numerical value is called for, enter a real number of -1 for not applicable responses.
2. If a "yes" or "no" response is called for, circle "Y" or "N". For not applicable responses write NA over the Y or N.
3. If the item is numeric and is not applied to the development in question, e.g., the development standards do not require problem reports to be filed against the requirements, or otherwise "not applicable" enter a negative (-1) or "NA".
4. The name of the system and/or module must be common across all worksheets.
5. A separate worksheet number 2b and 3 must be filled out for each module.

6. A maximum of 180 characters may be entered into the comments section of each worksheet.

The data elements are defined below in outline form corresponding to the worksheets. Each worksheet is self-contained and uses no raw data beyond itself.

WORKSHEET 1

This worksheet will use the requirements document(s) and an analyst will manually fill out the worksheets.

I. COMPLETENESS

1. Self explanatory
2. Self explanatory
3. Self explanatory
4. Refers to I-3
5. Refers to I-1
6. Refers to I-1
7. Refers to I-3
8. Refers to I-4 and I-7
9. Self explanatory
10. Self explanatory, often not applicable. If NA, enter -1.
11. Refers to I-10. If I-10 is NA this must be -1.

II. PRECISION

- 1-6. All are yes or no responses

III. SECURITY

- 1-5. All are yes or no responses

IV. HUMAN INTERFACE

- 1-6. All are yes or no responses

V. PERFORMANCE

1. May be stated in either numbers or prose

VI. SYSTEM INTERFACES

1. yes or no

VII. INSPECTOR'S COMMENTS

- Free format, limited to 180 characters

WORKSHEET 2a

This worksheet is used to evaluate the design documents at the system level and an inspector will manually fill out one for the system design.

I. COMPLETENESS

1. Mandatory yes or no
2. Self explanatory

- 3-5. Refer to I-2
- 6. May be NA. If NA, enter -1.
- 7-27. If I-6 is NA, all must be -1.

II. PRECISION

- 1-2. Mandatory yes or no
- 3. Requires a positive integer other than zero
- 4-5. Zero or positive integer
- 6-7. Mandatory yes or no

III. STRUCTURE

- 1-2. Mandatory yes or no
- 3. Requires a positive integer other than zero
- 4. Zero or positive integer
- 5. Mandatory yes or no

IV. OPTIMIZATION

- 1-5. Mandatory yes or no
- 6. Yes, No, or NA
- 7. Mandatory yes or no
- 8. Positive integer or zero
- 9-11. If IV-8 is zero, must be zero or -1.

V. SECURITY

- 1. Mandatory yes or no
- 2. Yes, No, or NA
- 3-4. Mandatory yes or no

VI. SYSTEM INTERFACES

- 1. Positive integer or zero
- 2. Mandatory yes or no
- 3. Yes or no. If VI-2 is No, must be No.
- 4. Positive integer excluding zero
- 5. Mandatory yes or no
- 6. If VI-5 is no, must be no
- 7. Positive Integer or zero

VII. HUMAN INTERFACE

- 1. Mandatory yes or no
- 2. Positive integer excluding zero
- 3-4. Either zeros or a unit figure. Be sure measurement units are same for 3 and 4.
- 5-6. Mandatory yes or no
- 7-8. Positive integer or zero
- 9-13. Mandatory yes or no
- 14-17. Positive integer or zero
- 18-23. Mandatory yes or no
- 24. Positive integer or zero
- 25-27. Mandatory yes or no

VIII. TESTING

- 1-10. Positive integers or zero
- 11. Mandatory yes or no

IX. DATA BASE

1-3. Positive integers or NA

X. INSPECTOR'S COMMENTS

Free format, limited to 180 characters.

WORKSHEET 2b

This worksheet is used to evaluate the design documents at the module level. An inspector will fill out one for each module design document(s).

I. COMPLETENESS

1. Mandatory yes or no

2. Positive integer or zero

3. Mandatory yes or no

4-5. Positive integers, zero, or NA. If NA, enter -1.

6-24. Positive integers, zero, or NA. If I-4 is NA, (-1) 6-24 is NA. All NA's are entered as -1.

II. PRECISION

1-9. Mandatory yes or no

III. STRUCTURE

1-4. Positive integers or zero

5-7. Mandatory yes or no

8-9. Positive integer excluding zero

IV. REFERENCES

1-4. Positive integers or zero

5-14. Mandatory yes or no

V. EXPANDABILITY

1-3. Mandatory yes or no

VI. OPTIMIZATION

1. Mandatory yes or no

2. Circle 1, 2, 3, or 4

3-5. Mandatory yes or no

VII. FUNCTIONAL CATEGORIZATION

Circle 1, 2, 3, 4, 5, or 6

VIII. CONSISTENCY

1-6. Mandatory yes or no

IX. INSPECTOR'S COMMENTS

Free format, limited to 180 characters.

WORKSHEET 3

This worksheet is used to evaluate the source code of each module. An inspector must enter the following items manually. The remainder are automatically gathered from the source code.

- I. STRUCTURE
 - 3, 14, 15, 16, and 18 Positive integer or zero
 - 19-20. Mandatory yes or no
- II. CONCISENESS
 - 1-4. Positive integer
- III. SELF-DESCRIPTIVENESS
 - 2. Positive integer or zero
 - 3-4. Mandatory yes or no
 - 5. Positive integer or zero
 - 6-7. Yes, no, or NA
 - 8. Positive integer or zero
 - 9-11. Mandatory yes or no
 - 12. Positive integer or zero
- IV. INPUT/OUTPUT
 - 3-6. Mandatory yes or no
- V. REFERENCES
 - 2-6. Positive integer or zero
 - 7-9. Mandatory yes or no
- VI. DATA
 - 1-5. Positive integer or zero
- VII. ERROR HANDLING
 - 1. Positive integer or zero
 - 2-4. Mandatory yes or no
- VIII. EFFICIENCY
 - 3-5. Positive integer or zero
- IX. PORTABILITY
 - 1. Mandatory yes or no
 - 2. Positive integer or zero
 - 3-4. Mandatory yes or no
- X. FLEXIBILITY
 - 1-4. Mandatory yes or no

XI. DYNAMIC MEASUREMENTS

- 1. Yes, no, or NA
- 2-3. Both items must have similar units of measurements or zero
- 4-7. Positive integers or zero. May be expressed in units of hundreds or thousands
- 8-9. Positive integer or zero.

XII. INSPECTOR'S COMMENTS

Free format, limited to 180 characters.

APPENDIX C

EXPLANATION OF METRICS

(Refs 17, 23)

EXPLANATIONS OF METRICS

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

Traceability

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

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

Completeness

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

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

- (1) Unambiguous references (input, function, output).

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

- (2) All data references defined, computed, or obtained from an external source.

Each data element is to have a specific origin. At the requirements level only major global data elements and a few specific local data elements may be available to be checked. The set of data elements available for completeness checking at the design level increases substantially and is to be complete at implementation.

- (3) All defined functions used.
A function which is defined but not used during a phase is either nonfunctional or a reference to it has been omitted.
- (4) All referenced functions defined.
A system is not complete at any phase if dummy functions are present or if functions have been referenced but not defined.
- (5) All conditions and processing defined for each decision point.
Each decision point is to have all of its conditions and alternative processing paths defined at each phase of the software development. The level of detail to which the conditions and alternative processing are described may vary but the important element is that all alternatives are described.
- (6) All defined and referenced calling sequence parameters agree.
For each interaction between modules, the full complement of defined parameters for the interface is to be used. A particular call to a module should not pass, for example, only five of the six defined parameters for that module.
- (7) All problem reports resolved.
At each phase in the development, problem reports are generated. Each is to be closed or a resolution indicated to ensure a complete product.

Consistency

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

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

(1) Standard Design Representation.

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

(2) Calling sequence conventions.

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

(3) Input/Output Conventions.

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

(4) Error Handling Conventions.

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

CS.2 Data Consistency Measure (Design and implementation at system level)

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

(1) Standard data usage representation.

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

(2) Naming Conventions.

Naming conventions for variables and modules are to be established and followed.

(3) Consistent Global Definitions.

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

Accuracy

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

- (1) Error analysis performed and budgeted to module (requirements phase only).

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

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

- (3) Sufficiency of Math Library (design phase only).

The accuracy of the math library routines utilized within the system is to be checked for consistency with the overall accuracy objectives.

- (4) Sufficiency of numerical methods (design and implementation phase).

The numerical methods utilized within the system are to be consistent with the accuracy objectives. They can be checked at design and implementation.

- (5) Execution outputs within tolerances (implementation phase only requiring execution).

A final measure during development testing is execution of modules and checking for accuracy of outputs.

Error Tolerance

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

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

- (1) Concurrent processing centrally controlled.

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

- (2) Errors fixable and processing continued.

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

- (3) When an error condition is detected, the condition is to be passed up to calling routine.

The decision of what to do about an error is to be made at a level where an affected module is controlled. This concept is built into the design and then implemented.

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

- (1) A definitive statement of requirement for error tolerance of input data.
The requirements specification must identify the error tolerance capabilities desired (requirements phase only).
- (2) Range of values (reasonableness) for items specified and checked (design and implementation phases only).
The attributes of each input item are to be checked for reasonableness. Examples are checking items if they must be numeric, alphabetic, positive or negative, of a certain length, nonzero, etc. These checks are to be specified at design and exist in code at implementation.
- (3) Conflicting requests and illegal combinations identified and checked (design and implementation phases only).
Checks to see if redundant input data agrees, if combinations of parameters are reasonable, and if requests are conflicting should be documented in the design and exist in the code at implementation.
- (4) All input is checked before processing begins (design and implementation phases only).
Input checking is not to stop at the first error encountered but to continue through all the input and then report all errors. Processing is not to start until the errors are reported and either corrections are made or a continue processing command is given.
- (5) Determination that all data is available prior to processing.
To avoid going through several processing steps before incomplete input data is discovered, checks for sufficiency of input data are to be made prior to the start of processing.

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

- (1) A definitive statement of requirement for recovery from computational failures (requirements phase only).

The requirement for this type error tolerance capability are to be stated during requirements phase.

- (2) Loop and multiple transfer index parameters range tested before use. (design and implementation phase only).

Range tests for loop indices and multiple transfers are to be specified at design and to exist in code at implementation.

- (3) Subscript checking (design and implementation phases only).

Checks for legal subscript values are to be specified at design and coded during implementation.

- (4) Critical output parameters reasonableness checked during processing (design and implementation phases only).

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

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

- (1) A definitive statement of requirements for recovery from hardware faults (requirements only).

The handling of hardware faults such as arithmetic faults, power failure, clock interrupts, etc., are to be specified during requirements phase.

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

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

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

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

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

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

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

Simplicity

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

- (1) Design organized in top down fashion.

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

(2) Module independence.

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

(3) Module processing not dependent on prior processing.

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

(4) Each module description includes input, output, processing, limitations.

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

(5) Each module has single entrance, single exit.

Determination of the number of modules that violate this rule at design and implementation can be made and is the basis for the metric.

(5) Size of data base.

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

(7) Compartmentalization of Data Base

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

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

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

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

(1) Module flow top to bottom.

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

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

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

(3) Jumps in and out of loops.

Loops within a module should have one entrance and one exit.

This measure is based on the number of loops which comply with this rule divided by the total number of loops.

(4) Loop index modified.

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

(5) Module is not self-modifying.

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

(6) Number of statement labels.

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

(7) Nesting level.

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

(8) Number of branches.

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

(9) Number of GOTO's.

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

(10) Variable mix in a module.

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

(11) Variable density.

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

Modularity

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

(1) Hierarchical Structure.

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

(2) Module Size Profile.

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

- (3) Controlling parameters defined by calling module.

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

- (4) Input data controlled by calling module.

See (3) above.

- (5) Output data provided to calling module.

See (3) above.

- (6) Control returned to calling module.

See (3) above.

- (7) Modules do not share temporary storage.

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

Generality

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

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

Expandability

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

- (1) Logical processing independent of storage specification/requirements. The logical processing of a module is to be independent of storage size, buffer space, or array sizes. The design provides for variable dimensions and dynamic array sizes to be defined parametrically. The metric is based on the number of modules containing hard-coded dimensions which do not exemplify this concept.
- (2) Percent of memory capacity uncommitted (implementation only). The amount of memory available for expansion is an important measure. This measure identifies the percent of available memory which has not been utilized in implementing the current system.

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

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

- (1) Input, processing, output functions are not mixed in a single function.

A module which performs I/O as well as processing is not as general as a module which simply accomplishes the processing. This measure is based on the number of modules that violate this concept at design and implementation.

- (2) Application and machine dependent functions are not mixed in a single module (implementation only).

Any references to machine dependent functions within a module lessens its generality. An example would be referencing the system clock for timing purposes. This measure is based on the number of machine dependent functions in a module.

- (3) Processing not data volume limited.

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

- (4) Processing not data value limited.

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

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

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

- (2) Modules table driven.

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

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

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

Instrumentation

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

- (1) Path coverage.

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

- 2) Input parameters boundary tested.

The other aspect of module testing involves testing the input

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

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

(1) Module interfaces tested.

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

(2) Performance requirements (timing and storage) coverage.

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

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

(1) Module Coverage.

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

(2) Identification of test inputs and outputs in summary form.

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

Self Descriptiveness

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

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

- (1) Modules have standard formatted prologue comments.

This information is extremely valuable to new personnel who have to work with the software after development, performing maintenance, testing, changes, etc. The measure at the system level is based on the number of modules which do not comply with a standard format or do not provide complete information.

- (2) Comments set off from code in uniform manner.

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

- (3) All transfers of control and destinations commented.

This form of comment aids in the understanding and ability to follow the logic of the module. The measure is based on the number of modules which do not comply.

- (4) All machine dependent code commented.

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

- (5) All non-standard HOL statements commented.

A similar explanation to 1) above is applicable here.

- (6) Attributes of all declared variables commented.

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

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

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

- (1) High Order Language used.

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

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

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

- (3) Source code logically blocked and indented.

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

- (4) One statement per line.

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

Execution Efficiency

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

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

- (1) Non-loop dependent computations kept out of loop.

Such practices as evaluating constants in a loop are to be avoided. This measure is based on the number of non-loop dependent statements

AD-A115 501

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/8 9/2
SOFTWARE QUALITY METRICS: A SOFTWARE MANAGEMENT MONITORING METH--ETC(U)
MAR 82 S J JARZOMBK

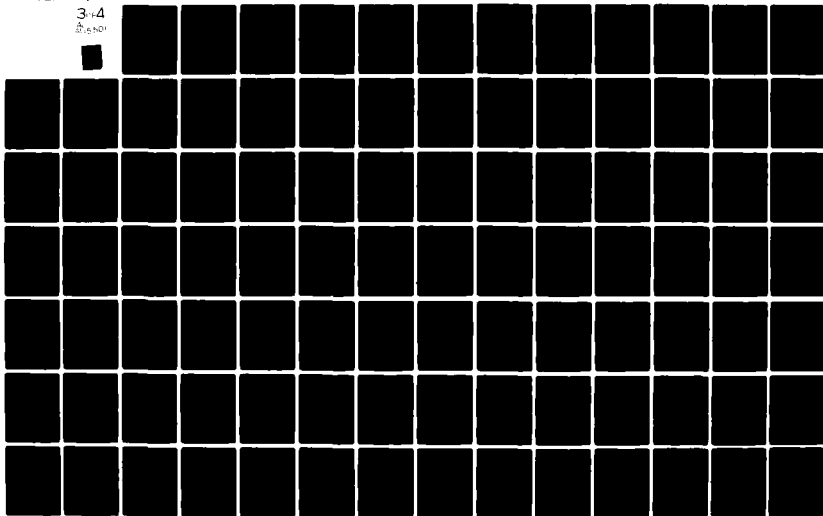
UNCLASSIFIED

AFIT/SCS/NA/82M-1

NL

3-4

5-10-1



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

- (2) Performance Optimizing Compiler/Assembly language used (implementation only).

This is a binary measure which identifies if a performance optimizing compiler was used (1) or if assembly language was used to accomplish performance optimization (1) or not (0).

- (3) Compound expressions defined once (implementation only).

Repeated compound expressions are to be avoided from an efficiency standpoint. This metric is based on the number of compound expressions which appear more than once.

- (4) Number of overlays.

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

- (5) Free of bit/byte packing/unpacking in loops.

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

- (6) Module linkages (implementation only, requires execution).
This measure essentially represents the inter-module communication overhead. The measure is based on the amount of execution time spent during module to module communication.
- (7) Operating System linkages (implementation only, requires execution).
This measure represents the module to OS communication overhead. The measure is based on the amount of execution time spent during module to OS communications.
- (8) Efficient Use of storage facility.
This measure represents an evaluation of the utility of the storage facility.

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

- (1) Data grouped for efficient processing.
The data utilized by any module is to be organized in the data base, buffers, arrays, etc., in a manner which facilitates efficient processing. The data organization during design and implementation is to be examined to provide this binary measure.
- (2) Variables initialized when declared (implementation only).
This measure is based on the number of variables used in a module which are not initialized when declared.

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

- (3) No mix-mode expressions (implementation only).

Processing overhead is consumed by mix-mode expressions which are otherwise unnecessary. This measure is based on the number of mix-mode expressions found in a module.

- (4) Common choice of units/types.

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

- (5) Data indexed or referenced for efficient processing.

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

Storage Efficiency

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

- (1) Storage Requirements allocated to design (design phase only).

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

(2) Virtual Storage Facilities Used.

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

(3) Common data defined only once (implementation only).

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

(4) Program Segmentation.

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

(5) Dynamic memory management used.

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

(6) Data packing used (implementation only).

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

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

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

Access Control

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

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

- (1) User I/O Access controls provided.

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

- (2) Data Base Access controls provided.

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

(3) Memory protection across tasks.

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

Access Audit

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

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

(1) Provisions for recording and reporting access.

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

(2) Provisions for immediate indication of access violation.

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

Operability

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

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

- (1) All steps of operation described.

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

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

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

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

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

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

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

- (5) Job set up and tear down procedures described (implementation only). The specific tasks involved in setting up a job and completing it are to be described. This is usually documented during the implementation phase when the final version of the system is fixed. This is a binary measure of the existence of that description.
- (6) Hard copy log of interactions maintained (design and implementation phases). This is a capability that must be planned for in design and coded during implementation. It assists in correcting operational errors, improving efficiency of operation, etc. This measure identifies whether it is considered in the design and implementation phases (1) or not (0).
- (7) Operator messages consistent and responses standard (design and implementation phases). This is a binary measure applied during design and implementation to insure that the interactions between the operator and the system are simple and consistent. Operator responses such as YES, NO, GO, STOP, are concise, simple, and can be consistently used throughout a system. Lengthy, differently formatted responses not only provide difficulty to the operator but also require complex error checking routines.

Training

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

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

- (2) Realistic simulated exercises provided (implementation only).
This is a binary measure of whether exercises which represent the operational environment, are developed during the implementation phase for use in training.
- (3) Sufficient 'help' and diagnostic information available on-line.
This is a binary measure of whether the capability to aid the operator in familiarization with the system has been designed and built into the system. Provision of a list of legal commands or a list of the sequential steps involved in a process are examples.

Communicativeness

CM.1 User Input Interface Measure (all three phases at system level).

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

- (1) Default values defined (design and implementation).
A method of minimizing the amount of input required is to provide defaults. This measure, applied during design and implementation, is based on the number of defaults allowed divided by the total number of input parameters.
- (2) Input formats uniform (design and implementation).
The greater the number of input formats there are the more difficult the system is to use. This measure is based on the total number of input formats.
- (3) Each input record self-identifying.
Input records which have self-identifying codes enhance the accuracy of user inputs. This measure is based on the number of input records that are not self identifying divided by the total number of input records. It is to be applied at design and implementation.

- (4) Input can be verified by user prior to execution (design and implementation).

The capability, displaying input upon request or echoing the input automatically, enables the user to check his inputs before processing. This is a measure of the existence of the design and implementation of this capability.

- (5) Input terminated by explicitly defined logical end of input (design and implementation).

The user should not have to provide a count of input cards. This is a binary measure of the design and implementation of this capability.

- (6) Provision for specifying input from different media.

The flexibility of input must be decided during the requirements analysis phase and followed through during design and implementation. This is a binary measure of the existence of the consideration of this capability during all three phases.

CM.2 User Output Interface Measure (all three phases at system level).

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

- (1) Selective Output Controls.

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

- (2) Outputs have unique descriptive user oriented labels (design and implementation only).

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

- (3) Outputs have user oriented units (design and implementation).
This is a binary measure which extends (2) above to the individual output items.
- (4) Uniform output labels (design and implementation).
This measure corresponds to CM.1 (2) above and is the inverse of the number of different output formats.
- (5) Logical groups of output separated for user examination (design and implementation).
Utilization of top of page, blank lines, lines of asterisks, etc., provide for easy identification of logically grouped output. This binary measure identifies if these techniques are used during design and implementation.
- (6) Relationship between error messages and outputs is unambiguous (design and implementation).
This is a binary measure applied during design and implementation which identifies if error messages will be directly related to the output.
- (7) Provision for redirecting output to different media.
This is a binary metric which identifies if consideration is given to the capability to redirect output to different media during requirements analysis, design, and implementation.

Software System Independence

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

(1) Dependence on Software System Utility programs.

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

(2) Common, standard subset of language used.

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

Machine Independence

MI.1 Machine Independence Measure (design and implementation at system level).

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

- (1) Programming language used available on other machines.
This is a binary measure identifying if the programming language used is available (1) on other machines or not (0). This means the same version and dialect of the language.
- (2) Free from input/output references.
Input and output references bind a module to the current machine configuration. Thus the fewer modules within a system that contain input and output references, the more localized the problem becomes when conversion is considered. This measure represents that fact and is based on the number of I/O references within a module. It is to be applied during design and implementation.
- (3) Code is independent of word and character size (implementation only).
Instructions or operations which are dependent on the word or character size of the machine are to be either avoided or parametric to facilitate use on another machine. This measure applied to the source during implementation is based on the number of modules which contain violations to the concept of independence of word and character size.
- (4) Data representation machine independent (implementation only).
The naming conventions (length) used are to be standard or compatible with other machines. This measure is based on the number of modules which contain variables which do not conform to standard data representations.

Communications Commonality

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

- (1) Definitive statement of requirements for communication with other systems (requirements only).

During the requirement phase, the communication requirements with other systems must be considered. This is a binary measure of the existence of this consideration.

- (2) Protocol standards established and followed.

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

- (3) Single module interface for input from another system.

The more modules which handle input from another system, the more difficult it is to interface with another system and the more likely it is to use standard protocols. This measure based on the inverse of the number of modules which handle input is to be applied to the design specification and source code.

- (4) Single module interface for output to another system

For similar reasons as (3) above this measure is the inverse of the number of output modules.

Data Commonality

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

- (1) Definitive statement for standard data representation for communications with other systems (requirements only).
This is a binary measure of the existence of consideration for standard data representation between systems which are to be interfaced. This must be addressed and measured in the requirements phase.
- (2) Translation standards among representations established and followed (design and implementation).
More than one translation from the standard data representations used for interfacing with other systems may exist within a system. Standards for these translations are to be established and followed. This binary measure identifies if the standards are established during design and followed during implementation.
- (3) Single module to perform each translation (design and implementation).
For similar reasons as CC.1 (3) and (4) above, this measure is the inverse of the maximum number of modules which perform a translation.

Conciseness

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

The observed length of a module is

$$N_0 = N_1 + N_2 \text{ where:}$$

$$N_1 = \text{total usage of all operators in a module}$$

$$N_2 = \text{total usage of all operands in a module}$$

The calculated length of a module is

$$N_c = n_1 \log_2 n_1 + n_2 \log_2 n_2 \text{ where:}$$

$$n_1 = \text{number of unique operators in a module}$$

$$n_2 = \text{number of unique operands in a module}$$

The metric is normalized as follows:

$$1 - \frac{|N_c - N_0|}{N_0} \text{ or,}$$

$$0 \text{ if } \frac{|N_c - N_0|}{N_0} \text{ greater than 1}$$

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

APPENDIX D

METRIC ALGORITHMS

(Refs 17, 23)

METRIC ALGORITHMS

The metric scores are automatically calculated by the AMT after the data is input from the metric worksheets. This appendix lists the algorithms used to compute the metric scores. They are listed according to SDLC phases and according to system level or module level application.

REQUIREMENTS - SYSTEM LEVEL

$$\begin{aligned} \text{CP.1} = & (\text{WS1.I.2} + ((\text{WS1.I.3} - \text{WS1.I.4})/\text{WS1.I.3}) + \\ & ((\text{WS1.I.1} - \text{WS1.I.5})/\text{WS1.I.1}) + ((\text{WS1.I.1} - \text{WS1.I.6})/\text{WS1.I.1}) + \\ & \text{WS1.I.9} + (\text{WS1.I.11}/\text{WS1.I.10}))/6 \end{aligned}$$

$$\text{AY.1} = (\text{WS1.II.1} + \text{WS1.II.2})/2$$

$$\text{ET.2} = (\text{WS1.II.3})$$

$$\text{ET.3} = (\text{WS1.II.4})$$

$$\text{ET.4} = (\text{WS1.II.5})$$

$$\text{ET.5} = (\text{WS1.II.6})$$

$$\text{AC.1} = (\text{WS1.III.1} + \text{WS1.III.2} + \text{WS1.III.3})/3$$

$$\text{AA.1} = (\text{WS1.III.4} + \text{WS1.III.5})/2$$

$$\text{OP.1} = (\text{WS1.IV.1} + \text{WS1.IV.2} + \text{WS1.IV.3})/3$$

$$\text{CM.1} = (\text{WS1.IV.4})$$

$$\text{CM.2} = (\text{WS1.IV.6} + \text{WS1.IV.5})/2$$

$$\text{CC.1} = (\text{WS1.IV.1})$$

$$\text{DC.1} = (\text{WS1.VI.2})$$

DESIGN - SYSTEM LEVEL

$$TR.1 = WS2a.I.1$$

$$CP.1 = (((WS2a.I.2 - WS2a.I.4)/WS2a.I.2) + ((WS2a.I.2 - WS2a.I.3)/WS2a.I.2) + (WS2a.I.2 - WS2a.I.5)/WS2a.I.2) + ((WS2a.I.6 - WS2a.I.7)/WS2a.I.6))/4$$

$$AY.1 = WS2a.II.1$$

$$ET.1 = (WS2a.II.2 + ((WS2a.II.4 + WS2a.II.5)/WS2a.II.3))/2$$

$$ET.4 = WS2a.II.6$$

$$ET.5 = WS2a.II.7$$

$$SI.1 = (WS2a.III.1 + (1/WS2a.IX.1) + (WS2a.IX.3/WS2a.IX.1))/3$$

If WS2a.IX.1 = 0 Set SI.1 to Value of WS2a.III.1

$$MO.2 = (1 - (WS2a.III.4/WS2a.III.2))$$

$$GE.1 = (WS2a.III.4/WS2a.III.2)$$

$$IN.1 = ((WS2a.VIII.4/WS2a.VIII.3) + (WS2a.VIII.2/WS2a.VIII.1))/2$$

$$IN.2 = ((WS2a.VIII.6/WS2a.VIII.5) + (WS2a.VIII.8/WS2a.VIII.7))/2$$

$$IN.3 = ((WS2a.VIII.10/WS2a.VIII.9) + WS2a.VIII.11)/2$$

$$EE.2 = ((1/WS2a.IV.8) + WS2a.VI.7 + WS2a.IV.4 + ((WS2a.IV.9 - WS2a.IV.10)/WS2a.IV.9))/4$$

DESIGN - SYSTEM LEVEL

$$EE.3 = WS2a.IV.6$$

$$SE.1 = (WS2a.IV.1 + WS2a.IV.2 + WS2a.IV.3)/3$$

$$AC.1 = (WS2a.V.1 + WS2a.V.2 + WS2a.V.3)/3$$

$$AA.1 = WS2a.V.4$$

$$OP.1 = (WS2a.VII.1 + WS2a.VII.9 + WS2a.VII.10 + WS2a.VII.6 + (1 - (WS2a.VII.8/WS2a.VII.7)))/5$$

$$TN.1 = WS2a.VII.13$$

$$CM.1 = ((WS2a.VII.16/WS2a.VII.15) + (1/WS2a.VII.14) + (1 - (WS2a.VII.17/WS2a.VII.15)) + WS2a.VII.18 + WS2a.VII.19 + WS2a.VII.20)/6$$

$$CM.2 = (WS2a.VII.21 + WS2a.VII.22 + WS2a.VII.23 + (1/WS2a.VII.24) + WS2a.VII.25 + WS2a.VII.26 + WS2a.VII.27)/7$$

$$CC.1 = (WS2a.VI.2 + WS2a.VI.3 + (1/WS2a.VI.4))/3$$

$$DC.1 = (WS2a.VI.5 + WS2a.VI.6 + 1/WS2a.VI.7)/3$$

IMPLEMENTATION - SYSTEM LEVEL

$$CP.1 = ((1 - (WS2a.I.4/WS2a.I.2)) + (1 - (WS2a.I.3/WS2a.I.2)) + (1/WS2a.I.5))/3$$

$$ET.1 = (WS2a.II.2 + ((WS2a.II.4 + WS2a.II.5)/WS2a.II.3))/2$$

$$ET.4 = WS2a.II.6$$

$$ET.5 = WS2a.II.7$$

$$SI.1 = ((WS2a.IX.2/WS2a.IX.1) + (1/WS2a.IX.1))/2$$

$$MO.2 = (1 - WS2a.III.4/WS2a.III.2)$$

$$GE.1 = (WS2a.III.4/WS2a.III.2)$$

$$IN.1 = ((WS2a.VIII.2/WS2a.VIII.1) + (WS2a.VIII.4/WS2a.VIII.3))/2$$

$$IN.2 = ((WS2a.VIII.6/WS2a.VIII.5) + (WS2a.VIII.8/WS2a.VIII.7))/2$$

$$IN.3 = ((WS2a.VIII.10/WS2a.VIII.9) + WS2a.VIII.11)/2$$

$$EE.2 = ((1/WS2a.IV.8) + ((WS2a.IV.10 - WS2a.IV.11)/WS2a.IV.9))/2$$

$$EE.3 = WS2a.IV.6$$

$$SE.1 = (WS2a.IV.2 + WS2a.IV.5 + (1 - (WS2a.IV.10/WS2a.IV.9)) + WS2a.IV.3)/4$$

$$AC.1 = (WS2a.V.1 + WS2a.V.2 + WS2a.V.3)/3$$

IMPLEMENTATION - SYSTEM LEVEL

$$AA.1 = WS2a.V.4$$

$$OP.1 = (WS2a.VII.1 + WS2a.VII.9 + WS2a.VII.10 + (1 - WS2a.VII.3/WS2a.VII.4) + WS2a.VII.5 + WS2a.VII.6 + (1 - (WS2a.VII.8/WS2a.VII.7)))/7$$

$$TN.1 = (WS2a.VII.11 + WS2a.VII.12 + WS2a.VII.13)/3$$

$$CM.1 = ((WS2a.VII.16/WS2a.VII.15 + (1/WS2a.VII.14) + (WS2a.VII.17/WS2a.VII.15) + WS2a.VII.18 + WS2a.VII.19 + WS2a.VII.20)/6$$

$$CM.2 = (WS2a.VII.21 + WS2a.VII.22 + WS2a.VII.23 + (1/WS2a.VII.24) + WS2a.VII.25 + WS2a.VII.26 + WS2a.VII.27)/7$$

$$CC.1 = ((1/WS2a.VI.1) + WS2a.VI.2 + WS2a.VI.3 + (1/WS2a.VI.4))/4$$

$$DC.1 = (WS2a.VI.5 + WS2a.VI.6 + (1/WS2a.VI.7))/3$$

DESIGN - MODULE LEVEL

$$CP.1 = (WS2b.I.1 + (1/WS2b.I.2) + WS2b.I.3 + (1 - WS2b.I.6/WS2b.I.4))/4$$

$$CS.1 = (WS2b.VIII.1 + WS2b.VIII.2 + WS2b.VIII.3 + WS2b.VIII.4)/4$$

$$CS.2 = (WS2b.VIII.5 + WS2b.VIII.6)/2$$

$$AY.1 = WS2b.II.2$$

$$ET.1 = WS2b.II.1$$

$$ET.2 = (WS2b.II.3 + WS2b.II.4 + WS2b.II.5 + WS2b.II.6)/4$$

$$ET.3 = (WS2b.II.7 + WS2b.II.8 + WS2b.II.9)/3$$

$$SI.1 = (WS2b.III.5 + WS2b.III.6)/2$$

$$SI.3 = 1/WS2b.III.1$$

$$MO.2 = ((WS2b.IV.4/WS2b.IV.3) + WS2b.IV.5 + WS2b.IV.6 + WS2b.IV.7 + WS2b.IV.8)/5$$

$$GE.2 = ((1 - WS2b.IV.9) + (1/WS2b.IV.10) + WS2b.IV.11 + WS2b.IV.12)/4$$

$$EX.1 = WS2b.V.1$$

$$EX.2 = (WS2b.V.2 + WS2b.V.3)/2$$

$$EE.1 = WS2b.VI.1$$

DESIGN - MODULE LEVEL

$$EE.2 = (WS2b.VI.3 + WS2b.VI.4)/2$$

$$EE.3 = WS2b.VI.5$$

$$SS.1 = ((1/WS2b.IV.1) + WS2b.IV.13)/2$$

$$MI.1 = (WS2b.IV.14 + 1/WS2b.IV.2)/2$$

IMPLEMENTATION - MODULE LEVEL

$$TR.1 = WS3.III.4$$

$$CP.1 = (WS2b.I.1 + 1/WS2b.I.2 + WS2b.I.3 + (1 - WS2b.I.6/WS2b.I.4))/4$$

$$CS.1 = (WS2b.VIII.2 + WS2b.VIII.3 + WS2b.VIII.4)/3$$

$$CS.2 = (WS2b.VIII.5 + 1/WS3.VI.3)/2$$

$$AY.1 = (WS2b.II.2 + WS3.XI.1)/2$$

$$ET.1 = WS3.VII.3$$

$$ET.2 = (WS3.IV.4 + WS3.IV.5 + WS3.IV.6)/3$$

$$ET.3 = ((1/WS3.VII.1) + WS3.VII.2 + WS3.VII.4)/3$$

IMPLEMENTATION - MODULE LEVEL

$$SI.1 = (WS2b.III.5 + WS2b.III.6 + WS2b.III.7 + \\ (1/(WS2b.III.8 + WS2b.III.9)))4$$

$$SI.3 = 1/WS3.I.10$$

$$SI.4 = (WS3.I.20 + (1 - (WS3.I.18/WS3.I.2 - WS3.I.4))) + \\ (1 - (WS3.I.15/WS3.I.14)) + (1 - (WS3.I.16/WS3.I.14)) + \\ (1 - (WS3.I.17/WS3.I.2)) + (1 - (WS3.I.6/(WS3.I.2 - WS3.I.4))) + \\ 1/WS3.I.9 + (1 - ((WS3.I.10 - WS3.I.11)/(WS3.I.2 - WS3.I.4))) + \\ (1 - ((WS3.I.12 - WS3.I.13)/(WS3.I.2 - WS3.I.4))) + \\ (WS3.VI.1/(WS3.VI.1 + WS3.VI.2)) + \\ (1 - ((WS3.VI.1 + WS3.VI.2)/(WS3.I.2 - WS3.I.4)))/11$$

$$MO.2 = (WS3.I.1 + WS3.V.5/WS3.V.3 + 1/WS3.V.4 + \\ 1/WS3.V.6 + WS3.V.7 + WS3.V.8 + WS3.V.9 + WS2b.IV.8)/8 \\ \text{For } WS3.I.1 \text{ } \_ 100 \text{ then } 0 \\ \text{For } WS3.I.1 \text{ } 100 \text{ then } 1$$

$$GE.2 = (WS2b.IV.9 + 1/WS3.IX.2 + WS3.X.2 + WS3.X3)/4 \\ \text{If } WS3.IX.2 = 0 \text{ Set 2nd term to } 1$$

$$EX.1 = (WS2b.V.1 + (WS3.XI.7/(WS3.XI.4 + WS3.XI.5 + WS3.XI.6 + WS3.XI.7)))/2$$

$$EX.2 = (WS3.X.4 + WS3.X.1 + (WS3.XI.5 - WS3.XI.4)/WS3.XI.5))/3$$

$$SD.1 = WS3.III.2/WS2.I.2 \\ \text{If } SD.1 \text{ } 1 \text{ Set to } 1$$

IMPLEMENTATION - MODULE LEVEL

$$SD.2 = (WS3.III.3 + WS3.III.11 + 1/WS3.III.5 + \\ WS3.III.6 + WS3.III.7 + 1/WS3.III.8 + WS3.III.10)/7$$

$$SD.3 = ((1 - WS3.I.3/WS3.I.1) + WS3.III.9 + WS3.III.11 + \\ (1 - WS3.III.12/WS3.I.1))/4$$

$$EE.2 = 1 - (WS3.VIII.3/WS3.I.14)$$

$$EE.3 = ((WS3.VIII.2/(WS3.VI.1 + WS3.VI.2)) + \\ (1 - WS3.VIII.1/(WS3.I.2 - WS3.I.4)) + \\ 1/WS3.VI.4 + WS2b.VI.5 + (1 - (WS3.XI.9/(WS2a.IX.1)))/5$$

$$SS.1 = ((WS3.V.2/(WS3.I.2 - WS3.I.4)) + WS2b.IV.13)/2$$

$$MI.1 = (WS2b.IV.14 + (1 - ((WS3.IV.1 + WS3.IV.2)/(WS3.I.2 - WS3.I.4))) + \\ WS3.IX.1 + \\ (1 - (WS3.IX.2/(WS3.I.2 - WS3.I.4))) + WS3.IX.3)/5$$

APPENDIX E

DEFINITION OF QUALITY ATTRIBUTES

DEFINITION OF QUALITY ATTRIBUTES

The software quality attributes (factors or criteria) contained in this appendix are found in the literature. The authors of the quoted or paraphrased definitions are indicated in parentheses.

ACCEPTABILITY

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

ACCESSIBILITY

- extent that software facilitates the selective use of its components (Boehm)
- those attributes of software that provide for an audit and control of the access of software and data (McCall)

ACCOUNTABILITY

- extent that code usage can be measured (Boehm)

ACCURACY

- the degree of exactness of the data contained in a product unit (Cho)
- where mathematically possible a routine should give an approximation that is as close as practicable to the full machine precision for whatever machine it is running on (Schonfelder)

- extent that its outputs are sufficiently precise to satisfy its intended use (Boehm)
- the mathematical calculations are correctly performed (Rubey)
- measure of the quality of freedom from error, degree of exactness possessed by an approximation or measurement (Gilb)

ADAPTABILITY

- how much time and effort are required to modify a software system (Kosy)
- measure of the ease with which a program can be altered to fit differing user images and system constraints (Poole)
- measure of the ease of extending the product, such as adding new user functions to the product (Myers)
- a measure of the effort required to modify a computer program to add, change or remove a function or use the computer program in a different environment (includes concepts of flexibility and portability) (SAMSO)
- relates to ability of software to operate inspite of unexpected input or external conditions (USA ISRAD)

AVAILABILITY

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

COMMUNICATIVENESS

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

COMPATABILITY

- measure of portability that can be expected of systems

when they are moved from one given environment to another (Gilb)

COMPLETENESS

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

COMPLEXITY

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

CONCISENESS

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

CONSISTENCY

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

CONVERTIBILITY

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

CORRECTNESS

- correctness of its description with respect to the objective of the software as specified by the semantic description of the linguistic level it defines (Dennis)
- the coding of a computer program module which properly

and completely implements selected overall system requirements (NSSC PATHWAY)

- relates to degree to which software is free from design and program defects (USA ISRAD)
- the program is logically correct (Rubey)

COST

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

Economy of operation - relates to cost of operating system

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

Economy of development - relates to cost of entire development cycle from identification of requirement to initial operation

- development and maintenance costs (Myers)
- implementation cost and operational cost (Gilb)

DOCUMENTATION

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

EFFICIENCY

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

expended (Gilb)

- the amount of computing resources and code required by a program to perform a function (McCall)
- reduction of overall cost - run time, operation, maintenance (Kernighan)
- extremely fast run time and efficient overlay capabilities (Richards)
- computation time and memory usage are optimized (Rubey)

EXPANDABILITY/FLEXIBILITY/AUGMENTABILITY

- attributes of software that provide for expansion of data storage requirements or computational functions (McCall)
- extent to which system can absorb workload increases and decreases which require modification (SADPR-85)
- the ability of a system to immediately handle different logical situations (Gilb)
- how easily the software modules comprising the system or subsystem can be rearranged to change the system's functions without modifying any of the modules (Kosy)
- ease of changing, expanding, or upgrading a program (Yourdon)
- the software modules must be usable in a variety of contexts (Culpepper)
- includes changeability; e.g., the ease of correction bugs, maintenance because of changing specifications, and portability to move to another system (Ledgard)
- extent that software easily accommodates expansions in data storage requirements or component computational functions (Boehm)
- attributes of software which allow quick response to changes in algorithms (Richards)
- ability to reuse the software and transfer it to another processor (includes reuse, adaptability, transferability and versatility of software) (Light)

EXPRESSION

- how a program is expressed determines in large measure

the intelligibility of the whole (Kernighan)

EXTENSIBILITY

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

GENERALITY/REUSABILITY

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

HUMAN FACTORS

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

INTEGRITY

- extent to which access to software or data by unauthorized persons can be controlled (McCall)

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

INTEROPERABILITY

- effort required to couple one system with another (McCall)
- how quickly and easily one software system can be coupled to another (Kosy)
- relates to how quickly and easily one software system can be coupled to another (USA ISRAD)

LEGIBILITY

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

MAINTAINABILITY

- ability to keep up with its intended use (Cho)
- measure of the effort and time required to fix bugs in the program (Myers)
- how easy it is to locate and correct errors found in operational use (Kosy)
- extent that the software facilitates updating to satisfy new requirements (Boehm)
- maintenance involves
 - (1) correction to heretofore latent bug

- (2) enhancements
- (3) expansion
- (4) major redesign (Lieblein)

- ease with which a change can be made due to
 - (1) bug during operation
 - (2) non-satisfaction of users requirements
 - (3) changing requirements
 - (4) obsolescence/upgrade of system (McCall)
- probability that a failed system will be restored to operable condition within a specified time (Gilb)

MANAGEABILITY

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

MODIFIABILITY

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

MODULARITY/STRUCTUREDNESS

- extent to which a system uses program constructs for better readability (Cho)

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

PERFORMANCE

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

PORTABILITY/TRANSFERABILITY

- programs must be readily transferable among different equipment configurations (Goodenough/Culpepper)
- degree of transportability is determined by number, extent, and complexity of changes, and hence the difficulty in implementing a software processor which can mechanically move a program, between a specified set of machines (Hague)
- machine - independence (Marshall)
- measure of the ease of moving a computer program from one computing environment to another (Meahy, Poole)
- how quickly and cheaply the software system can be converted to perform the same functions using different equipment (Kosy)
- an appropriate environment can be provided on most

computers (Goos)

- extent that it can be operated easily and well on computer configurations other than its current one (Boehm)
- moving software from one computer (environment) to another (Lieblein)
- measure of the effort required to install a program on another machine, another operating system, or a different configuration of the same machine (Wulf)
- external (black box) form, fit, and function characteristics of a program module which permit its use as a building block in several computer programs (NSSC PATHWAY)
- relates to how quickly and easily a software system can be transferred from one hardware system to another (USA ISRAD)
- ease of conversion of a system from one environment to another; the relative conversion cost for a given conversion method (Gilb)
- effort required to transfer a program from one hardware configuration and/or software system environment to another (McCall)

PRECISION

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

PRIVACY

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

RELIABILITY

- software's ability to perform what it is supposed to do under defined conditions (Cho)
- includes correctness, testing for errors, and error

tolerance (Ledgard)

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

REPAIRABILITY

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

ROBUSTNESS

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

validity (Kernighan)

SECURITY

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

SELF-CONTAINEDNESS

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

SELF-DESCRIPTIVENESS/CLARITY

- those attributes of software that provide explanation of the implementation of a function (McCall)
- measure of how clear a program is, i.e., how easy it is to read, understand, and use (Ledgard)
- refers to the ease with which the program (and its documentation) can be understood by humans (Wulf)
- extent that it contains enough information for a reader to determine its objectives, assumptions, constraints, inputs, outputs, components, and status (Boehm)

SERVICEABILITY

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

STABILITY

- the "ripple effect" or how many modules have to be changed when you make a change (Myers)
- measure of the lack of perceivable change in a system

in spite of the occurrence in the environment which would normally be expected to cause a change (Gilb)

TESTABILITY

- the ease of checking the quality of system output through use of messages (Cho)
- instrumentation and debugging aids (Liskov)
- minimize testing costs (Yourdon)
- provision of facilities in the design of programs which are essential to testing complex structures (Edwards)
- extent that software facilitates the establishment of acceptance criteria and supports evaluation of its performance (Boehm)
- a measure of the effort required to exercise the computer program to see how well it performs in a given environment and if it actively solves the problem it was supposed to solve (SAMSO)
- effort required to test a program to insure it performs its intended function (McCall)
- measure of our ability to test software (Light)

TIME

- two major categories of time:

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

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

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

TOLERANCE

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

UNDERSTANDABILITY

- meaningful variable names, brief comments, traceable module interface and data flow (Cho)
- ease with which the implementation can be understood (Richards)
- reduced complexity, reduced redundancy, clear documentation/notation (Goodenough)
- extent that the purpose of the product is clear to the evaluator (Boehm)
- documentation remains current (Whipple)
- the program is intelligible (Rubey)

UNIFORMITY

- module should be usable uniformly (Goodenough)

USABILITY/OPERABILITY

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

VALIDITY

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

APPENDIX F

SOFTWARE SYSTEM DEVELOPMENT LIFE CYCLE FOR AFLC/LM

(Ref 89)

ADS DEVELOPMENT

USING DOD ADS DOCUMENTATION STANDARD
7935.1-5 AND AFR 300-12, 300-15; PREPARED BY
HQ AFEC/LME JUNE 1981

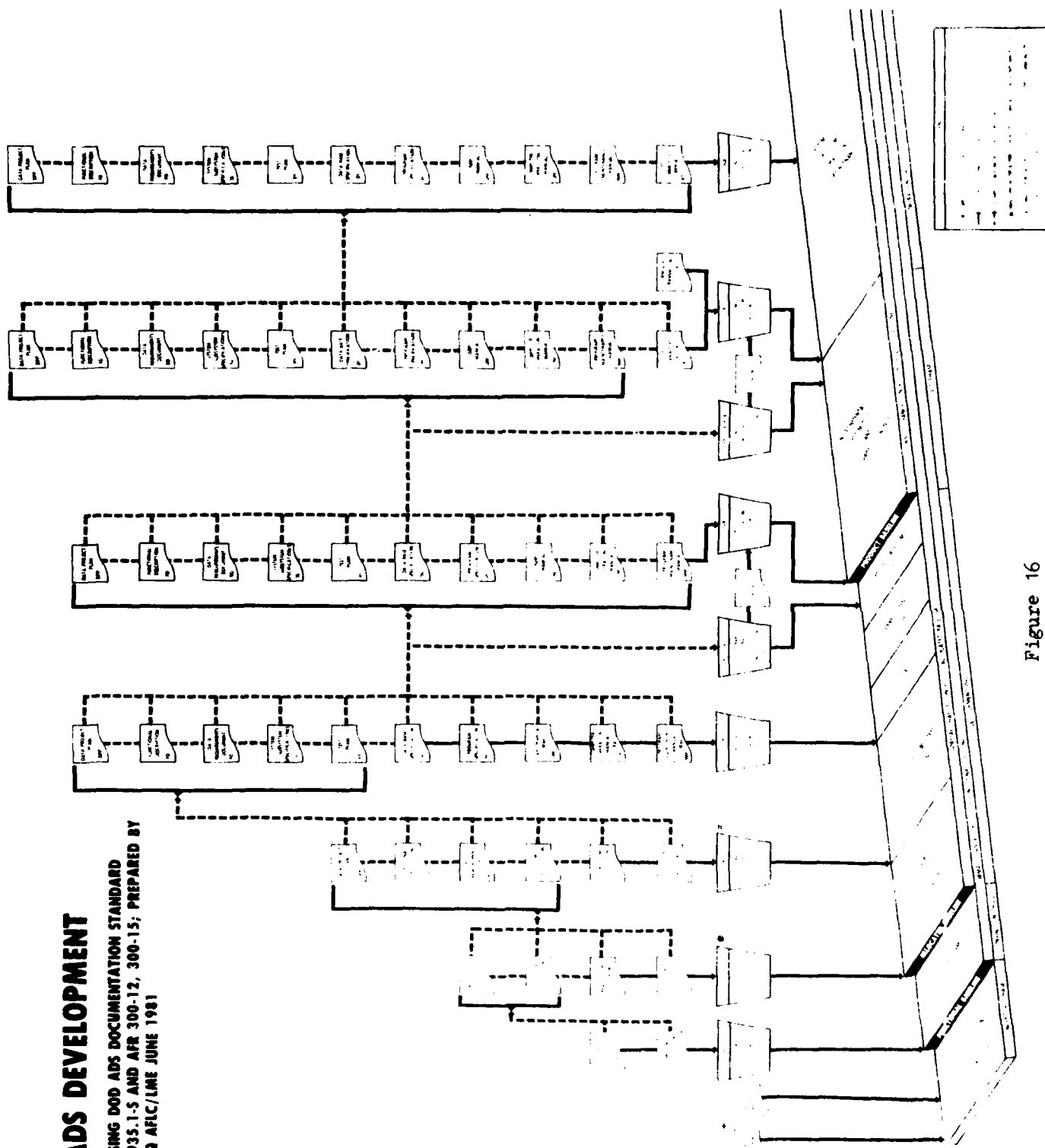


Figure 16

Reviews

Purpose of Reviews. The purpose of a AFR 300-15 review is to give a project manager (PM) and deputy project manager (DPM) a means of assessing and verifying the degree of completion of tasks related to milestones and a means of controlling resources. When a review has been completed the PM/DPM can give management the information on which to base a decision to continue, redirect, or end a project. While all reviews have the common purpose of "assessing and verifying," they also have more specific unique purposes. An explanation of the unique purpose of each review, as derived from AF regulations and experience follows:

a. System Requirements Review (SRR).

(1) To make sure the requirements documented in the Functional Description (FD) by the functional OPR (proponent), which are to be satisfied by a data system(s), are clear, complete, correct and consistent with those in the validated and approved Data Automation Requirement (DAR).

(2) To make sure the functional OPR and data automation OPR have a mutual understanding of the requirements in the FD which are to be satisfied by the data system(s).

(3) To make sure the Data Project Plan (DPP), which is the plan for the management, development, and implementation of the system(s), is clear, complete, correct, and consistent with the direction in the Data Project Directive (DPD).

(4) To form the basis for establishing the Functional Baseline.

b. System Design Review (SDR).

(1) To make sure the functions documented in the System/Subsystem Specification (SS), which are needed to satisfy each requirement documented in the FD, have been assigned (allocated) to specific programs (components) in the data system(s).

(2) To make sure the functions to be carried out by each program in the data system(s) will satisfy the requirements of the functional OPR documented in the FD.

(3) To make sure all manual (human), automatic data processing equipment (ADPE), and data system (software) interfaces have been named, defined, and negotiated.

(4) To form the basis for establishing the Allocated Baseline.

c. Preliminary Design Review (PDR).

(1) To make sure the general (macro) technical design of each program (component) in the data system(s) is acceptable and will satisfy the requirements of the functional OPR documented in the FD.

(2) To make sure the programs that make up the data system(s) will satisfy the requirements of the functional OPR documented in the FD.

d. Critical Design Review (CDR).

(1) To make sure the detailed technical design of each program in the data system(s) will satisfy its intended purpose (satisfy the requirements of the functional OPR) prior to programming (coding).

(2) To make sure all support documents (Users Manual (UM), Operations Manual (OM), Maintenance Manual (MM), and Test and Implementation Plan (PT)) are clear, complete (draft), correct, and consistent with the data(s) to be programmed.

e. Product Verification Review (PVR).

(1) To make sure the product (Computer Program Configuration Item (CPCI)) that has been programmed and tested by data automation satisfies all functional and technical requirements.

(2) To make sure all associated documentation agrees with the product that has been programmed.

(3) To make sure the product is ready for functional user testing (validation testing) and the preparations for the Test Phase have been completed.

(4) To form the basis for establishing the Product Baseline.

f. System Validation Review (SVR).

(1) To confirm that system performance, as proven during validation testing, meets the functional OPR (proponents) requirements documented in the FD and DAR(s).

(2) To make sure the system is technically acceptable for operation in a day-to-day environment.

(3) To make sure all necessary arrangements have been made for implementation and operational support of the system.

Preparing for Reviews. Preparation is the single most important factor contributing to the success of any AFR 300-15 review. It's the responsibility of the PM/DPM to see that preparations for a review are complete before committing external project resources to it. A description of the activities usually associated with preparing for any review are named and described in the following subparagraphs. It must be emphasized, these are activities usually associated with preparing for any review. Since each project varies in scope, complexity, and direction received, it would be impossible to name all possible activities associated with each review for all possible projects. Realizing this, each PM/DPM must fully understand the specific purpose of each review, must comply with direction and directives in tailoring the review to his/her project, and must make sure all prerequisite activities are completed prior to a review. The following are the usual activities.

a. Identifying Items. The PM/DPM must first name the items that will be assessed and verified for completeness prior to and/or during a particular review. To make this determination, the PM/DPM should rely on AFR 300-15, Chapter 3, the projects Data Project Directive (DPD) and Data Project Plan (DPP), as well as Volume 1, Part A of this handbook. A list of the reviews and the items usually reviewed follow:

(1) SRP - DAR(s), DPD(s), DPP, FD, and all amendments.

(2) SDR - Same as above plus the SS (Sections 1, 2, & 3), FT (if prepared), configuration management records, and previous review minutes.

(3) Same as above plus the completed SS and a draft of the FT.

(4) CDR - Same as above except final FT (less Section 2), PSs, DS (if prepared) and a draft of the MM, OM, and UM.

(5) PVR - Same as above except final PT, MM, OM, and UM.

(6) SVR - Same as above plus the RT.

b. Determining Acceptability. The PM/DPM is responsible for making sure the item(s) to be reviewed are prepared in accordance with governing directives before sending it to people for review. The PM/DPM also has an obligation to make sure, to the best of his/her ability, the item(s) is a quality product that will serve its intended purpose. To make sure a quality item will be delivered the PM/DPM should, at key points, monitor the completion of tasks associated with the preparation of the item. This can be done by scheduling walk-throughs, audits, briefings, interim reports, etc. As an example, the PM/DPM should review the results of the functional systems analysis tasks, before Sections 2 and 3 of the FD are started/completed. By scheduling and conducting such internal reviews the PM/DPM can preclude the delivery of items that have not been prepared according to direction and do not serve the purpose for which they were intended.

c. Identifying Participants. The identification of people to review items and/or attend reviews is critical to the success of a project. The PM/DPM, working with functional and development OPRs/OCRs, should find and gain commitment of functionally/technically qualified people to review each item and/or attend each review. The PM/DPM should look on those selected as constructive critics and not as adversaries. The following subparagraphs give more specific information to assist the PM/DPM in the selection of participants:

(1) Those in attendance should be responsible for (PM/DPM, OPR/OCR), or be able to make a material contribution (based on purpose of the review) to the success of the project.

(2) It's not necessary for representatives of all functional user organizations (as opposed to functional OPR/OCR), interfacing data automation organizations (as opposed to the development OPR/OCR), or representatives of other interfacing organizations (SAC, AFDSDC, AFAFC, etc.) to be in attendance. However, if needed, the PM/DPM or functional/development OPR/OCR can ask representatives of user/data automation/interfacing organizations to review documentation and supply Design Problems Reports (DPRs)/coordination. The PM/DPM or functional/development OPR may also ask representatives from these same organizations to act as advisors at reviews when their attendance will materially contribute to the success of the project.

(3) When the project manager expects controversy or plans changes in the project which will need review and approval (RCB, PMR, CMR, HQ AF, etc.), representatives from organizations responsible for giving or coordinating on project direction (HQ AF/LE/AC, HQ AFLC/AC/SC/XR, AFDSEC, AFSA, etc.) should be in attendance. The knowledge gains by these representatives will be instrumental in enabling them to help in resolving controversy and in forming an organizational position on project direction (continue, redirect, end).

(4) The purpose of each review is different, so participation can be different. As an example, the functional OPR may plan for selected functional users to attend the SRR, but not the SDR. Likewise, the development OPR may plan for all lead programmers to attend the CDR, but not the SRR. Further, the data system support (DSS) manager may be in attendance at the SRR to answer questions about project direction in the DPD, but may not attend the CDR because it is not a "baseline" review.

(5) Each project manager, when preparing the DPP for his/her project, should describe in Section VI of the Configuration Management Plan (CMP), which is an appendix to the DPP, plans for conducting project reviews and audits. These plans, in addition to identifying the reviews/audits to be conducted, will name who the reviewing authorities will be. When the DPP is coordinated, organizations involved in the project may suggest (with explanations) adding/dropping review participants as proper.

d. Planning Format. The format for a review meeting can vary depending on the project, the review being conducted, and the preferences of the PM/DPM. The PM/DPM should always develop and document, in the proposed agenda, the format of the meeting. In developing the format the PM/DPM should always remember the specific purpose of a review and see the purpose is served. A general outline for reviews which has proven successful and keeps the PM/DPM in control follows.

(1) Opening Remarks - Given by PM/DPM to explain purpose, format, procedures, and administrative matters.

(2) Introductions - Given by PM/DPM to introduce all/key participants and their roles, relationship to project, and responsibilities.

(3) Project Background

(a) Functional - Given by functional OPR/OCR(s) to explain what is to be accomplished.

(b) Data Automation - Given by development OPR/OCR(s) to explain how data automation will be used to satisfy the requirements of the OPR/OCR(s).

(c) Project Management - Given by PM/DPM to explain history of project to date.

(4) Items to be Reviewed - Given by PM/DPM to name items to be reviewed, their purposes, and their relationships.

(5) Review of Items - Led by PM/DPM to assess and verify the completion of tasks related to milestones.

(6) Summary of Open DPR(s) - Identified by PM/DPM to inform participants of actions to be completed before review is completed.

(7) Plans for Completing Items - Given by PM/DPM to inform participants of what actions will be taken when to update/"baseline" review items.

(8) Plans for Coordination - Given by PM/DPM to inform participants of what coordination is needed and when it will be carried out to complete the review.

(9) Future Plans - Given by PM/DPM to inform participants of near term project tasks.

(10) Closing - Given by PM/DPM.

Preparing Agenda. The PM/DPM is responsible for preparing an agenda which outlines the format for each review. In addition to naming the major subject and primary participants, the agenda should include location(s), date(s), start-stop times, break/meal times and other pertinent planning information. The final agenda should reflect the results of a coordinated planning effort which most effectively makes use of Air Force time and funds. (Reference Atch 2 for sample agenda)

i. Sending Material. The PM/DPM is responsible for assembling and sending, with a letter of transmittal, the proposed agenda and material to be reviewed. The letter should be addressed to all participating people/organizations. It should identify what is being transmitted and explain what is expected of participants prior to attending the review. As an example the PM/DPM could ask the agenda be reviewed and any proposed changes called in by a certain date. The PM/DPM could also ask all DPRs be sent no later than 5 workdays prior to the review. The PM/DPM should consider the amount of material to be reviewed and allow adequate time for a thorough review of the material by the participants and enough time for PM/DPM review of the DPRs.

g. Reviewing Design Problem Reports. The PM/DPM should receive all DPRs, except those prepared at the review, prior to the review meeting. Each DPR should be reviewed and a coordinated position developed with the OPR/OCR(s) prior to the review. Those DPRs needing work to resolve that will extend beyond the dates of the SRR should have an estimated completion (suspense) date. The DPRs should be assigned control numbers and entered into the DPR Log. The PM/DPM should determine, based on those DPRs received, how best to address/present them at the review. As an example, all administrative DPRs (editorial discrepancies) could be aggregated together and addressed all together by deliverable rather than individually. The PM/DPM may decide to modify the review agenda, based on the volume, content, and validity of the DPRs received.

h. Planning Support. The PM/DPM should make sure all administrative support details are taken care of prior to a review. These details include, as an example:

- (1) Secretarial support.
- (2) Lodging and transportation for participants.
- (3) Responsibilities of project office people prior/during SRR.
- (4) Supplies.
- (5) Room(s) for meeting.

Conducting Review Meetings. Next to adequately preparing for a review, properly conducting the review meeting is the most important factor contributing to its success. AFR 300-15 places the responsibility for chairing all reviews on the PM or his/her designee. The chairperson, for the purpose of this handbook, is considered to be the PM/DPM, and so it is his/her responsibility to conduct all review meetings. To chair a review, the PM/DPM must understand the purpose of it and must plan to see its purpose is satisfied. Said another way, "before leading, you must know where you are going and

have decided on how best to get there." The PM/DPM can best satisfy the purpose of a review by adhering to the "coordinated" agenda he/she should have prepared. The following subparagraphs give an example of opening a review, conducting a review of selected items, and closing the review. These subparagraphs were prepared based on the format in paragraph d above which is assumed to have been incorporated into the agenda:

a. Opening a Meeting. The PM/DPM opens the review by introducing himself/herself. This introduction is followed with an explanation of the purpose of the review. This explanation should be more than quoting AFR 300-15. As an example; "The purpose of this SRR is to review, complete, approve, and place under change control the OPRs and OCRs ADS requirements defined in the FD. I want the considered opinions of those present as to whether the stated requirements are correct, complete, testable, and communicate to the end user and development OPR/OCR(s) a clear statement of the operational capability to be developed. I also want to review certain near-term plans so everyone is familiar with their respective responsibilities and tasks." The PM/DPM next gives an explanation of how he/she intends to satisfy the stated purpose of the review. As an example, "I will first review the FD, section by section. While reviewing a section, I will discuss each DPR against that section and its proposed disposition." The PM/DPM would then give an explanation of the procedural and administrative details. As an example, "If there is a need to prepare DPRs during this review, I will identify and discuss the need and if valid the DPR form will be prepared by the originator, assigned for action, and given a control number."

Following these opening remarks, the PM/DPM should introduce all participants or key individuals, such as the functional OPR/OCR(s), development OPR/OCR(s) and people representing other Commands or services. As the individuals are being introduced, their relationship and responsibility to the project should be explained. As an example, "Mr A from the Strategic Air Command is responsible for acquiring B and developing C and interfacing it with the D part of this project."

Following his/her introduction the functional OPR/OCR(s) should be asked to give an explanation of what the LMS requirements and benefits are in general terms. Such as: "This requirement for ADP support is necessary to enable the XYZ organization to - - - . If successful it will save \$X and decrease response time to - - - ." This functional background explanation should be followed by an explanation by the development of how data automation will satisfy the requirements in the FD. For example: "the requirements in the FD do not need any special equipment or software, so we will use the XYZ as configured to satisfy all requirements." The ADP background explanation should then be followed by a brief background of the project by the PM/DPM. As an example, "The DAR was approved in March and the DPD was received in April. The project office was formed in April and the DPP completed in June. People from the ADP organization and the functional OPR/OCR(s) organization(s) completed their systems analysis in August. Subsequently, the draft FD was completed and forwarded to each of you in December." The PM/DPM would next name the items to be reviewed and their relationship to each other. He/she might say, "The DPP will be reviewed in context with the direction in the DPD. We received only 1 minor DPR against the DPP." Following the agenda, which should orient all participants to the project and give them an understanding of what will be done during the review, the PM/DPM begins the review.

Conducting a Review. Following the opening of the meeting, the PM/DPM begins the review of the first item. As mentioned before, the PM/DPM must tailor the review to his/her project. If the project is very large and complex and involves large numbers of people in the review meeting, the review should be more structured so the PM/DPM can remain in control. If the opposite is true, the meeting can be more informal. In either

case the PM/DPM must realize that he/she is in charge and is responsible for making sure the purpose of having the review is met. The following paragraphs address each of the six AFR 300-15 reviews and gives examples of how a PM/DPM could conduct each review meeting. Within the subparagraphs are points the PM/DPM could choose to bring up concerning the purpose of parts of the items being reviewed and relationships of parts within an item and between items. Further explanations on the items (DOD 7935.1-S documents) to be reviewed will be contained in a later edition of this volume of the handbook.

a. System Requirements Review (SRR). The SRR, for purposes of this handbook, begins with the PM/DPM identifying the DAR, DPD, DPP, and FD as the items to be reviewed. He/she also points out that DPRs will be addressed, when applicable, to the part of the item being reviewed.

The DAR and all amendments to it should be briefly summarized by the PM/DPM. He/she might state, "each of you have a copy of the original DAR. There has been 1 amendment to it which added the requirement to do A. This additive requirement added \$B in cost, C man-hours and D additional days to E task. The benefit is estimated to be \$F per year. These changes did not exceed the 15%/120 day thresholds where a management review is required. The certified Economic Analysis (EA) is still valid and represents the anticipated costs for this project. The last document to be reviewed will be the FD and all requirements therein emanate from this DAR and 1 amendment." After the discussions, if any, the DPD is reviewed.

The PM/DPM should briefly summarize the direction in the DPD and any amendment which is pertinent to the review. This could include direction such as: what items must be completed in the Conceptual Phase, who is responsible for producing a particular item, what reviews must be held, what level of approval is required if thresholds are breached, etc. After reviewing the relevant direction in the DPD, the DPP is reviewed. In some cases the PM/DPM may wish to summarize the DPP first.

The PM/DPM should discuss with review participants the major milestones shown in the DPP. Following this brief review the near term tasks and their OPR/OCS(s) should be reviewed. If there are schedule, cost, or resource changes expected they should be addressed. The PM/DPM should also explain how changes to the Functional Baseline affecting cost, schedule, or performance must be coordinated and approved. The PM/DPM should also explain his/her plans to add other appendices to the DPP and the procedures for making and coordinating changes to those already prepared. Sometimes the review of the DPP, because of already planned changes, would be better served by reviewing it last. If so, the agenda could be prepared showing the DPP will be reviewed following Section 5 of the FD.

The next and most critical document to be reviewed is the FD. The FD should be thoroughly reviewed section-by-section to make sure all attendees understand and agree with the contents. The PM/DPM should point out that the acceptance of the FD as a document which adequately defines the functional baseline is critical to the success of the entire project. Without a defined and controlled baseline to work from, the accomplishment of the ADS design, the preparation of the User Manual (UM) and the preparation of the Test Plan (PT) become an impossible task. The PM/DPM should recognize and emphasize the importance of certain paragraphs within sections of the FD. As an example, the functional OPR/OCR(s) should make sure an adequate analysis of the impact of the changes on the functional user organization has been done and the results clearly documented in paragraph 2.6.1. Likewise, the ADP OPR/OCR(s) should carefully analyze the timing requirements (paragraph 3.1.2) of each requirement (paragraph 3.1) and make sure the proposed environment described in Section 4 is capable of meeting

these requirements. If not, the known constraints or limitations should be identified in paragraph 2.7. The PM/DPM should also recognize and point out that particular attention must be paid to the contents of Section 3, since they are the basis for the contents of follow-on documents. As an example, the functions identified in paragraph 3.2 are further analyzed and detailed in the System/Subsystem Specification (SS), paragraph 2.2. The functions in the SS are allocated to programs and then further analyzed and detailed in the Program Specifications, paragraph 2.2. Paragraphs 4.1.1 and 4.1.2 of the PT are likewise dependent on specific and testable requirements and functions being identified in paragraphs 3.1 and 3.2 of the FD. The PM/DPM should use visual aids such as charts, diagrams, etc., to assist in the review of the FD when feasible.

After completing the review of each section of the FD, and of each DPR associated to each section, the review itself is considered complete. The PM/DPM now closes the review meeting.

b. System Design Review (SDR). As with the SRR, the SDR begins with the PM/DPM naming the first item for review. Experience has shown the PM/DPM should first attempt to familiarize all participants with the current Functional Baseline (FBL). This can be carried out by the PM/DPM briefly going over the SRR minutes and then identifying and reviewing any changes to the FBL since the SRR was concluded. This would include new DARs/DPDs, DAR/DPD amendments, Class I Baseline Change Requests (BCRs) against the FBL (FD), and changes in costs and benefits. This brief review serves to give all participants a common understanding of the FBL.

Before beginning the review of the next item, the System Subsystem Specification (SS), the PM/DPM should explain whether the OPR/OCR(s) requirements will be satisfied by one or more Computer Program Configuration Items (CPCIs). As pointed out in AFR 300-15, if there are multiple CPCIs each is required to successfully complete a SDR. Basically what this means to the PM/DPM is: if the system to be designed will consist of and be managed as a single entity (CPCI)--then a System Specification (SS) will define the design and will be reviewed at a SDR. If the system will consist of several manageable entities or the development and/or implementation will be phased--then the design of each CPCI will be defined in separate Subsystem Specifications each having a separate SDR. DOD 7935.1-S states, "If individual Subsystem Specifications are prepared, they may at some point be bound together to form a System Specification or a separate System Specification may be written." When multiple SSs are being prepared, the PM/DPM must so advise the reviewers so they are aware of what requirements in the FD will be satisfied by a CPCI and how each CPCI relates to the others in the overall system. For the purpose of this handbook a single CPCI will be assumed.

After attempting to set up this common point of reference for the FBL, the SS should be reviewed against it. That is, the SS should be reviewed against the Functional Baseline defined by the FD. The SS should be reviewed section by section and all DPRs against a section addressed while that section is being reviewed. The PM/DPM should ask the development OPR to give an overview of the proposed automated data system (ADS). This could be done from the charts required to satisfy paragraph 2.1 of the SS. The PM/DPM should next point out to the participants that the System/Subsystems Functions documented in paragraph 2.2 maintain a direct relationship to both the requirements and functions in the FD, paragraphs 3.1 and 3.2 respectively. Again, this could be illustrated by a chart showing the direct relationship of SS functions to FD functions and requirements (reference attachment 3). The PM/DPM must also, as required by AFLC Supplement I to DOD 7935.1-S, name what functions have been allocated to what programs (components) and where this allocation is documented. This relationship could

also be illustrated by matrix chart (reference attachment 4). The PM/DPM should next ask the development OPR to explain the contents of Section 3 which defines the ADP environment that will support the OPR/OCR(s) requirements.

Following the review of Sections 1 through 3 of the SS and all associated DPRs, the PM/DPM should name the next item for review. A similar review, section by section with DPRs, should be carried out against this item and all others, if feasible.

After reviewing the final item, the PM/DPM should present the results of any simulations of the proposed design in the SS or any other evidence not previously presented. The PM/DPM should entertain any new DPRs, discussions or questions concerning the SDR items. After discussions, if any, the PM/DPM should advise the participants the review of the items is completed and close the SDR meeting.

c. Preliminary Design Review (PDR). Before discussing the meeting, it is important to explain several inconsistencies in AFR 300-15 regarding PDRs. Following this discussion will be an explanation of the inconsistencies and of what AFLC/SC recommends to eliminate these inconsistencies.

AFR 300-15, figure 1-2 identifies the SS and Subsystem Specifications (SSs) as being separate items that are reviewed at the SDR and PDR respectively. However, DOD 7935.1-S, paragraph 2.4.3 clearly points out that either a SS or SSs will be prepared, not both. Further, AFR 300-15, figure 2-1, doesn't identify the SSs as being separate and distinct items. It shows a SS or SSs being placed under change control after a successful SDR. This position is consistent with the statement in AFR 300-15, paragraph 1-9a, but is contradicted by paragraph 3-7a(2). These are but a few of the inconsistencies surrounding the PDR and SS/SSs in AFR 300-15. To eliminate debate and meet the intent of DOD 7935.1-S and AFR 300-15, and also stay consistent with the way ADSs evolve, SC takes the following position. If there are multiple CPCIs in a ADS, each of their designs will be documented in separate Subsystem Specifications and be subjected to SDRs and PDRs. Conversely, if there is only one CPCI, its design will be documented in a System Specification and reviewed at a SDR and PDR. If the PM/DPM wants the design of the ADS system, which is defined in Sections 2 and 3 of the SS/SSs assessed and verified, this can be done at a SDR, without the finite details for each program being documented in Section 4. In large systems this could save considerable rework in Section 4 if the design of the ADS is found to be faulty. This is not to say the details in Section 4 are not available, they just are not available in finished form as Section 4 requires. Once having successfully completed a SDR, and established an Allocated Baseline (ABL), the agreed on controlled design (macro) of the ADS can now be defined (documented) at the program level in Section 4 of the SS/SSs. This level of detail, which logically follows the design of the ADS system or subsystem, can now be assessed and verified at a PDR. Following the PDR, each program documented in Section 4 will be designed at the lowest level and documented in a Program Specification. If the PM/DPM decides the ADS design has little chance of being found faulty, he/she can conduct a combined SDR/PDR at which time a complete (Sections 1 through 4) System/Subsystem Specification is reviewed. In summary AFLC/SC believes the logical evolution of a ADS system or subsystem is documented first in either an SS or in SSs not both. The ADS level design is documented in Sections 2 and 3, which is reviewed at the SDR, followed by the documentation of the program level (macro) design in Section 4, which is reviewed at the PDR.

Following the opening of the review, the PM/DPM reviews the SDR minutes and summarizes previous changes/additions to the FBL and Allocated Baseline (ABL). After any discussions, the review of Section 4 of the SS is begun. Section 4 should be reviewed

in context with the ABL and FBL. That is, the participants should assess and verify whether the macro level design of each program will enable the functions carried out by it to meet the OPR/OCRs requirements. Further, each program must be designed according to applicable standards and be able to accomplish the functions allocated within the technical limitations of the environment defined in Section 3. Just as the name infers, it is a preliminary design review of each component (program) in the ADS system or subsystem.

The last item to review is a draft of the Test and Implementation Plan (PT). Draft in this context means a PT in reviewable form, excluding the results of development testing required in Section 2. The draft PT is reviewed at this time to assess and verify the ADS and programs as designed are capable of being tested to demonstrate the OPR/OCR(s) requirements have been met. If not, either the ADS design, program design or the tests must be redesigned. The PM/DPM should explain the purpose of the PT, as explained in Section 1 and also explain that the tests specified must relate to the requirements and functions identified in the FD.

After completing the review of the SS, PT, and any associated DPRs, the PM/DPM should close the meeting.

d. Critical Design Review (CDR). The PM/DPM begins the review by identifying the first item to be reviewed. As with each review, the PM/DPM should attempt to bring all participants up-to-date as to the FBL and ABL. As explained previously, this is done by briefly discussing the PDR minutes and all approved changes made to the system since the last review. The PM/DPM will find the configuration management (CM) status accounting records can be useful in accomplishing this update.

Following the update, the PM/DPM names the Program Specification (PS) that will be reviewed. The PM/DPM may decide, in the case of large systems and/or when no DPRs have been received against a PS, that all PSs will not be reviewed at the meeting. If the PM/DPM decides all PSs will not be reviewed, he/she should explain why certain ones will and others will not. The PSs to be reviewed should be reviewed against the SS(s), which names the functions to be carried out by each program and its inputs, outputs, data bases, interfaces (internal/external). As explained by DOD 7935.1-S, pages 3-33, through 3-40, the definition of each program must relate to specific parts of the SS(s) and/or FD. Each PS reviewed must be assessed and verified as to its ability to meet the OPR/OCR(s) requirement, to its conformance to standards, and to its technical correctness. When the PSs are approved, they will be the basis for each programmer to code from. This is not to say that in certain cases where risk is low or techniques, algorithms, etc., require verification, that coding has not already begun or completed for selected programs. If this has been done the PM/DPM should so advise and use the results to aid in the review.

When the review of the proposed technical design of all or critical programs has been completed, the system support documents should be reviewed. These include drafts of the Users Manual (UM), Operations Manual (OM), and although considered by AFR 300-12 to be an internal document, the Maintenance Manual (MM), and a completed Test Plan (PT). These documents should be reviewed in context with the FD, SS(s), and PS(s). They must be reviewed to decide if they are correct and consistent given the ADS design solution specified in the SS, PS(s), and RD and DS if prepared. The ultimate objective of course is to make sure the ADS to be coded can be used by the functional user, operated by operations personnel, tested by an independent test team, and maintained by the ADP maintenance staff. If design deficiencies are noted during this review, the system/program design must be corrected and baseline documents changed accordingly.

After completing the review of all items the PM/DPM should close the meeting.

Product Verification Review (PVR). The PM/DPM should open the PVR with a brief summary of the results of the preliminary Physical and Functional Configuration Audits and the result of development testing. This should be followed with a discussion of the changes to the FBL and ABL that have taken place since the CDR.

Following these discussions, the audit report is reviewed and the items previously reviewed at the CDR are reviewed in their final form. The PM/DPM should explain to all participants, that once approved the support documents will be used by all people, functional and ADP, during the Test Phase. Therefore, each and every document should be reviewed as if it were ready for full operational use. The PM/DPM should also point out that subsequent to their review and approval, the PS(s) which define the Product Baseline will be placed under configuration control (Class I/II BCR).

After completing the review and addressing all DPRs the meeting is closed.

1. System Validation Review (SVR). This review, which is conducted at the end of the Test Phase, is conducted to assess and verify the systems acceptability for operational use. To accomplish this and terminate the development project, the PM/DPM must get the functional OPR/OCR(s) to certify the system that has been developed. To accomplish this, the PM/DPM should review the Test Analysis Report (RT) prepared by the independent test director. This report should be reviewed to decide if the tests conducted provide evidence that the functional OPR/OCR(s) requirements specified in the FD have been satisfied. Additionally, the results from the final FCA and PCA should be reviewed to decide if a quality product has been developed that meets all functional requirements and technical standards. Following these reviews, the PM/DPM should name and review the planned disposition of all outstanding DPR(s), System Problem Reports (SPRs) (prepared during validation testing), and BCRs. Following their review, the PM/DPM should identify to the representative of the ADP maintenance organization all of the items that will be turned over and the planned turnover date(s).

After the above has been accomplished the meeting should be closed.

g. Final Operational Evaluation (FOE). This evaluation (review), which is sometimes needed, is an Operational Phase review and is not discussed in AFR 300-15. The description of this review is described in AFR 300-12, paragraph 4-5(6). As AFR 300-12 explains, the FOE "is a review of the operational data system". For a multi-site system, this review will take place when the system is operational at a representative set of planned sites. The installation at the remaining sites then would be considered to be the implementation of an already operational system."

Since there is nothing more definitive concerning this review or the organization responsible for its accomplishment, it will not be discussed further in this handbook. The PM/DPM, if tasked by the DPD to do a FOE, should ask for specific instructions concerning its purpose, items to be reviewed, reports to be prepared, etc.

Closing a Meeting. In closing a review meeting, the PM/DPM should try to wrap up all loose ends. This gives participants an understanding of what actions are still required to successfully conclude a review, when they will be carried out, and who is responsible for their accomplishment. For example, the PM/DPM should announce when meeting minutes will be sent and name all DPRs that need further work to resolve. For each DPR he/she should name the responsible person/organization and the expected completion date. He/she should also name the actions still needed to set up a baseline and when

they will be completed, e.g., complete DPRs, update FD, forward FD to key participants with AFLC 406 for signature, conduct CCB meeting. Following these discussions, the PM/DPM should reiterate the major tasks between this review and the next review. Accomplishing this, the PM/DPM should then thank all participants for assisting him/her and dismiss the participants. After the meeting, the PM/DPM must now complete those actions necessary to successfully end the review.

Concluding Reviews. After a review meeting, the PM/DPM must take certain actions to successfully end a review and if necessary establish a baseline. These actions, which are specified in AFR 300-15 and AFLC Supplement 1, are discussed in the following paragraph.

The first concern of the PM/DPM, after a meeting, is to make sure all responsibilities are assigned and understood. Work should already be in progress toward resolving open DPRs, and the meeting minutes should be nearing completion, if secretarial help was available during the meeting. The PM/DPM should have a letter of transmittal ready to transmit the updated DOD 7935.1-S documents, AFLC 406, and minutes to meeting participants (Reference Atch 5). A CCB meeting (to recommend acceptance of a document for baselining) should be scheduled and prepared for as well as a AFLCR 400-18 Program Manager Review (PMR). The PM/DPM should also make sure the required CM logs are up-to-date and procedures have been set up to maintain the integrity of a baseline.

While all of the actions to conclude a review seem trivial, it must be remembered there are also a thousand and one other things taking place concurrently, relative to the project. So the PM/DPM should recognize and be ready to quickly conclude a review and have a defined controllable baseline to work from. This is not to say, rush to completion. Rather, it is saying be aware of all of the activities needed to end a successful meeting and make sure they are planned for and are carried out in an orderly and quick manner.

AUDITS

Purpose of Audits. Audits of a CPCI are conducted at two points in its development to determine whether the CPCI conforms to specifications and standards. On completion of an audit, the PM/DPM will receive an audit report citing the findings and recommendations of the audit team. A more detailed explanation of the purpose of each type audit follows:

a. Functional Configuration Audit (FCA). This audit is conducted to validate whether a CPCI's actual performance complies with the SS and meets the OPR/OCR(s) requirements specified in the FD.

b. Physical Configuration Audit (PCA). This audit is conducted to validate whether a CPCI agrees with its technical documentation, conforms to quality assurance policies and procedures, and adheres to applicable standards.

Preparing for Audits. The PM/DPM must get ready for, as noted previously, audits at two points in the development cycle of each CPCI. The first audit, which is entitled, "Preliminary Functional and Physical Configuration Audit," is conducted lead-time away from the scheduled PVR for a CPCI. The second audit is entitled, "Final Functional and Physical Configuration Audit," and is conducted lead-time away from the scheduled SVR for a CPCI.

To prepare for an audit, the PM/DPM must first notify the audit director, named in the DPD, of the particulars concerning the CPCI to be audited. This notification is then followed with the delivery of the items to be audited (reference AFR 300-15, Chapter 3, Section C for contents of notification and identification of the items to be audited).

Conducting Audits. Conducting audits are not the responsibility of the PM/DPM. However, the PM/DPM is very much interested in their results because they are a key element in determining whether the CPCI is ready for validation testing and later for operational use. The section of AFR 300-15 referenced in the preceding paragraph shows what will be done during a FCA and PCA. This section of AFR 300-15 also points out the difference between a preliminary and final configuration audit.

Concluding Audits. Both audits are concluded when the audit director prepares minutes (report) of the audit, citing the findings and recommendations of the audit team. These reports are given to the PM/DPM for use in the PVR and SVR. The PM/DPM should review these reports very carefully before actually conducting a PVR or SVR. If the findings indicate major system discrepancies exist, the PM/DPM must then determine if the PVR and SVR should be delayed. If major discrepancies do exist and the PM/DPM decides the PVR/SVR will be conducted prior to their correction, he/she should be ready to discuss their impact on validation testing/operation at the PVR/SVR/PMR.

I. DOCUMENTS NECESSARY TO BEGIN THE CONCEPTUAL PHASE

A. Required System Capability (RSC)

1. Prepared in accordance with (IAW) AFLCR 400-5.
2. Prepared by organization having the requirement (OPR).
3. Validated by AFLC/XRB, IAW AFLCR 400-5, attachment 2.
4. Prioritized by AFLC/XRB, IAW AFLCR 400-5, attachment 3.
5. Approved by AFLC/XRB or AF/LE.
6. Entered in AFLC Capabilities Plan (process, priority, etc.) by AFLC/XRB.
7. Entered in AFLC LMS Action Plan (objective, schedule, resources, etc.) by AFLC/SCC.

B. PROJECTED ADP REQUIREMENT (PAR)

1. Prepared IAW AFLC/ACOR Data Call Letter.
2. Prepared by organization having the requirement.
3. Presented to AFLC Management System Panel (MSP) by organization having equipment.
4. Approved and prioritized by MSP.
5. Entered into AFLC Major Command ADP Plan (MCAP) by AFLC/ACD.
6. If manpower is required, a 602 package is needed at the same time.

C. MANPOWER CHANGE REQUEST (MCR)

1. AF Form 602, prepared by Servicing Manpower Evaluation Team (MET) in cooperation with organization having requirement, IAW AFM 26-1, Chapter 9.
2. Authenticated by HQ AFLC/DPQ and sent to HQ USAF/PRM.

D. PRELIMINARY EVALUATION NOTICE (PEN)

1. Prepared by the organization having requirement IAW AFLC Supplement 1 to AFR 300-12.
2. Sent to AFLC/ACD for evaluation.
3. Evaluated by potential data automation development organization.
4. Returned to originating organization (via ACD) for use in preparing DAR.

E. DATA AUTOMATION REQUIREMENT (DAR)

1. Prepared IAW AFR 300-12, attachment 1 and AFLC Supplement 1.
2. Prepared by organization having the requirement.
3. Validated by AFLC/XRB IAW AFLCR 400-5, attachment 2.
4. Prioritized by AFLC/XRB IAW AFLCR 400-5, attachment 3.
5. Scheduled by AFLC/ACD.
6. Approved by AFLC/AC or AF/AC & LE or SAF/FM.
7. The following sequence should be followed FS, EA, DAR.

F. ECONOMIC ANALYSIS (EA)

1. Prepared IAW AFR 300-12, Chapter 3, Section B and AFLC Supplement 1, AFR 178-1 and AFLC Supplement 1.
2. Prepared by organization having the requirement (EA needed when estimated development man-years are five or more).
3. Certified by AFLC/ACM or AF/ACM.
4. Included (when needed) as attachment to DAR, IAW AFR 300-12, attachment 1 and attachments 8 thru 11.
5. Actual costs tracked (when needed) IAW AFR 300-12, Chapter 3, Section C, attachments 12 thru 14.
6. Administrative Personnel assigned to a project office must be costed to the project.

G. FEASIBILITY STUDY (FS)

1. Prepared IAW AFLC Supplement 1, attachment 3, to AFR 300-12.
2. Prepared by organization having the requirement (FS needed when estimated man-years are 10 or more).
3. Included (when needed as attachment to DAR, IAW AFR 300-12, attachment 1.

H. PRIVACY ACT STATEMENT

1. Prepared IAW AFR 300-12; Chapter 2, paragraph 2-10.
2. Prepared by organization having the requirement.
3. Included in DAR IAW AFLC Supplement 1, attachment 6 to AFR 300-12.

I. ADP AND TELECOMMUNICATIONS REQUIREMENTS CHECKLIST

1. Prepared IAW AFR 300-12, Chapter 5, 6, or 9.
2. Prepared by organization having the requirement (Checklist prepared when a Delegation of Procurement Authority (DPA) is needed).
3. Certified by AFLC/AC, AF/ACD, or SAF/FM.
4. Included (when needed) as attachment to DAR, IAW AFR 300-12, attachment 1 and 23.
5. All documentation supporting the checklist must be kept on file.

J. SITE PREPARATION REQUIREMENTS

1. Prepared IAW AFR 300-12, Chapter 2, paragraph 2-11 and AFLC Supplement 1.
2. Prepared by organization having the requirement.
3. Included in DAR, IAW AFLC Supplement 1, attachment 6 to AFR 300-12.

K. AF/ATC TRAINING REQUIREMENTS STATEMENT

1. Prepared IAW AFR 300-12, Chapter 2, paragraph 2-8.
2. Prepared by organization having the requirement.
3. Notification of training requirement (if applicable) to Air Training Command (ATC).
4. Included in DAR, IAW AFLC Supplement 1, attachment 6, to AFR 300-12.

L. TELECOMMUNICATIONS REQUIREMENTS STATEMENT

1. Prepared IAW AFR 300-12, Chapter 2, paragraph 2-9.
2. Prepared by organization having the requirement.
3. Requirement coordinated through HQ AFLC/DC with Air Force Communications Command (AFCC).
4. Included in DAR, IAW AFLC Supplement 1, attachment 6, to AFR 300-12.

M. NEW START/STOP REQUEST

1. Prepared IAW AFM 26-1, Chapter 1 and AFLC Supplement 1.
2. Prepared by organization having the requirement.
3. Sent to AF/MPMX (with copy to DAR) by AFLC/DP.

N. ENVIRONMENTAL ASSESSMENT (FORMAL/INFORMAL)/STATEMENT

1. Prepared IAW AFR 19-2 and AFLC Supplement 1.
2. Prepared by organization having the requirement.
3. Certified by proponent of requirement.
4. Accompanies DAR through decision making process to HQ USAF.

O. STATEMENT OF WORK (SOW)

1. Prepared IAW AFR 300-12, Chapter 6, paragraph 6-2, and attachment 23.
2. Prepared by organization requiring contractor support.
3. Sent with DAR and approved by HQ AFLC/ACD or DAR approval authority.
4. Sent to servicing contracting office or GSA (with GSA Form 2068) by HQ AFLC/ACD.

II. DOCUMENTS/ACTIONS NECESSARY TO COMPLETE THE CONCEPTUAL PHASE

A. DATA PROJECT DIRECTIVE (DPD)

1. DPD received from DAR approval authority (supplemented by ADP Single Manager as appropriate).
2. Prepared IAW AFR 300-12, Chapter 2, paragraph 2-6, attachment 2, and AFLC Supplement 1.
3. Directed actions to be taken by project participants.

B. DATA PROJECT PLAN (DPP)

1. Prepared IAW AFR 300-12, attachment 3 and AFLC Supplement 1.
2. OPR - Project Management Office (PMO).
3. OCR(s) - Organizations participating in supporting project.
4. Approved by DAR approval authority.
5. Describes actions to achieve project performance, schedule, and cost; objectives specified in DPD.

C. FUNCTIONAL DESCRIPTION (FD)

1. Prepared (when needed) IAW AFR 300-12, attachment 26.
2. OPR - Project Management Office.
3. OCR(s) - Organizations participating in/supporting project.
4. Baseline by project manager subsequent to System Requirements Review (SRR).
5. Defines system requirements (after system analysis) to be satisfied.
6. DODS 7935.1-S should be read prior to starting preparation of the FD.

D. SYSTEM REQUIREMENTS REVIEW (SRR)

1. Conducted IAW AFR 300-15, Chapter 3 and AFLC Supplement 1.
2. Chaired by project manager of his/her designee.
3. Attended by representatives of participating/support (project) organizations.
4. Conducted to review/finalize contents of FD and make sure all project participants/supporters have mutual understanding of the ADS requirements. In addition the DAR, DPD, and DPP must also be reviewed.

E. CONFIGURATION CONTROL BOARD (CCB) MEETING

1. Conducted IAW AFR 300-15, Chapter 2 and AFLC Supplement 1.
2. Chaired by project manager or his/her designee.
3. Attended by members specified in DPD or DPP.
4. Conducted to approved Configuration Control Directive (CCD) (AFLC Form 406) requesting initial baselining or to review/classify requested changes to established baselines.

F. PROGRAM MANAGER REVIEW (PMR)

1. Conducted IAW AFLCR 400-18.
2. Chaired by AFLC/SC (LMS program manager).
3. Project presentations given by project manager (or his/her designee).
4. Attended by project manager, deputy project manager and representatives from participating/supporting organizations.
5. Conducted after completion of a major milestone (or at least every 6 months) to assess project progress and give direction (continue/change/terminate) per AFLC Form 1298.

G. ADPE/PERFORMANCE SPECIFICATION

1. Prepared IAW AFR 300-12, Chapter 9.
2. OPR - Project Management Office.
3. OCR - Organizations participating in/supporting project.
4. Approved by approval authority shown in AFR 300-2, attachment 2 (contact HQ AFLC/ACD for verification).
5. Delegation of Procurement Authority requested from GSA (if needed) by approval authority.
6. If approval authority is HQ USAF/AC), RFP prepared by AFCAC (reference AFR 300-12, Chapter 9 and attachment 18).
7. If approval authority is HQ USAF/ACD, RFP prepared by servicing contracting office (reference AFR 300-12, Chapter 9 and attachment 15).

H. STATEMENT OF WORK (SOW) - See entry I.O.

I. DATA AUTOMATION REQUIREMENT/AMENDMENT

1. Prepared, when needed, based on criteria in AFR 300-12, Chapter 2, paragraph 2-5b.
2. See entry I.E. for remaining actions/responsibilities.

III. ACTIONS NECESSARY TO BEGIN THE DEFINITION PHASE

A. DATA PROJECT DIRECTIVE/UPDATE/AMENDMENT

1. Prepared, when needed, based on criteria in AFR 300-12, Chapter 2, paragraph 2-6.
2. See entry II.A for remaining actions/responsibilities.

B. DATA AUTOMATION REQUIREMENT/AMENDMENT

1. Prepare baseline change request forms as needed.
2. See entry II.E and II.I for remaining actions/responsibilities.

C. STATEMENT OF WORK - SEE ENTRY I.O.

IV. ACTIONS NECESSARY TO COMPLETE THE DEFINITION PHASE

A. DATA PROJECT PLAN UPDATE

1. PMO updates those parts of the DPP that need updating as a result of changes in project scope (DAR/DPD amendments, etc.), schedule, resources, benefits, etc.
2. See entry II.B and AFLCR 400-18 for remaining actions/responsibilities.

B. FUNCTIONAL DESCRIPTION CHANGES

1. PMO changes the FD as a result of Baseline Change Requests, which are prompted by DAR(S)/DAR amendments, SPR(s), etc. Update Configuration Management Status Accounting Logs as appropriate.
2. See entry II.C and II.E for remaining actions/responsibilities.

C. SYSTEM/SUBSYSTEM SPECIFICATION (SS)

1. Prepared (when needed) IAW DOD Standard 7935.1-S.
2. OPR - Project Management Office.
3. OCR - Organizations participating in/support project (ADS design accomplished by ADP personnel).
4. Baselined after the System Design Review (SDR).
5. Defines the ADS design (system/subsystem/programs) further defines the ADS functions, and identifies (allocates) the functions from the FD to each program.

D. DATA REQUIREMENTS DOCUMENT (RD)

1. Prepared IAW DOD Standard 7935.1-S.
2. OPR - Project Management Office.
3. OCR - Organizations participating in/supporting project.
4. Baselined after the System Design Review.
5. Defines and lists data elements and defines users data collection requirements.

E. SYSTEM DESIGN REVIEW (SDR)

1. Conducted IAW AFR 300-15, Chapter 3 and AFLC Supplement 1.
2. Chaired by project manager of his/her designee.
3. Attended by representatives of participating/supporting organizations.

4. Conducted to review/finalize contents of SS(s) and RD (when applicable) and make sure proposed ADS design will satisfy user's needs as specified within the FD.

F. CONFIGURATION CONTROL BOARD - See entry II.E.

G. PROGRAM MANAGER REVIEW - See entry II.F.

V. ACTIONS NECESSARY TO BEGIN THE DEVELOPMENT PHASE

- A. DATA PROJECT DIRECTIVE/UPDATE/AMENDMENT - See entry II.A.**
- B. DATA AUTOMATION REQUIREMENT/AMENDMENT - See entry III.B.**
- C. STATEMENT OF WORK - See entry I.O.**

VI. ACTIONS NECESSARY TO COMPLETE THE DEVELOPMENT PHASE

A. DATA PROJECT PLAN UPDATE - See entry IV.A.

B. FUNCTIONAL DESCRIPTION CHANGE - See entry IV.B.

C. SYSTEM/SUBSYSTEM SPECIFICATION/CHANGE

1. PMO changes the SS as a result of Baseline Change Requests, which are prompted by DAR(S)/DAR amendments, FD baseline changes, etc.
2. See entry IV.C. and II.E. for remaining actions/responsibilities.

D. DATA REQUIREMENTS DOCUMENT CHANGE

1. PMO changes the RD as required.
2. See entry IV.D. and II.E. for remaining actions/responsibilities.

E. DATA BASE SPECIFICATION (DS)

1. Prepared IAW DOD Standard 7935.1-S.
2. OPR - Project Management Office.
3. OCR - Organizations participating in/supporting project (Data base design done by ADP personnel).
4. Baselined after the Product Verification Review.
5. Describes the Storage allocation and data base (tape, disk, etc.) organization.

F. PROGRAM SPECIFICATION(S) (PS)

1. Prepared (when needed) IAW DOD Standard 7935.1-S.
2. OPR - Program Management Office.
3. OCR - Organizations participating in/supporting project (program design done by ADP personnel).
4. Baselined after the Produce Verification Review.
5. Defines the design of each program in the ADS/SS and identifies the function(s) performed by the program(s).

G. TEST PLAN (PT)

1. Prepared (when needed) IAW DOD Standard 7935.1-S and AFLC Supplement I.
2. OPR - Project Management Office.

3. OCR - Organizations participating in/supporting project (reference AFR 300-15, Chapter 5 and AFLC Supplement 1 for more specifics).
 4. Baseline after the Product Verification Review.
 5. Defines the test program (plan) to be conducted by an independent test group.
- H. MAINTENANCE MANUAL (MM)
1. Prepared IAW DOD Standard 7935.1-5, AFLC Supplement 1 and AFR 300-15, AFLC Supplement 1.
 2. OPR - Project Management Office.
 3. OCR - Organizations participating in/supporting project.
 4. Baseline after the Product Verification Review.
 5. Describes computer programs in a detailed technical presentation to assist maintenance programmer(s).
- I. USER'S MANUAL (UM)
1. Prepared IAW DOD Standard 7935.1-5 and AFLC Supplement 1.
 2. OPR - Project Management Office.
 3. OCR - Organizations participating in/supporting project (prepared by the ADS "user" organization(s)).
 4. Published/managed IAW AFR 5-4 and AFR 5-1.
 5. Provides the "user" organization(s) management and staff with information and instructions for managing and using the ADS.
- J. COMPUTER OPERATION MANUAL (OM)
1. Prepared IAW DOD Standard 7935.1-5 and AFLC Supplement 1.
 2. OPR - Project Management Office.
 3. OCR - Organizations participating in/supporting project.
 4. Baseline after to the Product Verification Review.
 5. Gives the information needed by computer operations personnel to operate the ADS.
- K. PRELIMINARY DESIGN REVIEW (PDR)
1. Conducted IAW AFR 300-15, Chapter 3 and AFLC Supplement 1.
 2. Chaired by project manager or his/her designee.

3. Attended by representatives of participating/supporting organizations.
4. Conducted to review completed SS(s), which specifies the design of each Computer Program Configuration Item (CPCI). Also to review a draft of the test plans contained in the Test Plan (TP).

L. CRITICAL DESIGN REVIEW (CDR)

1. Conducted IAW AFR 300-15, Chapter 3 and AFLC Supplement 1.
2. Chaired by project manager or his/her designee.
3. Attended by representatives of participating/supporting (project) organizations.
4. Conducted to review the design of each program within the CPCI, usually prior to coding. Other supporting documents are also reviewed (reference AFR 300-15 and AFLC Supplement 1) to make sure the proposed design solution will meet all requirements.

M. PRELIMINARY FUNCTIONAL/PHYSICAL CONFIGURATION AUDIT

1. Conducted IAW AFR 300-15, Chapter 3 and AFLC Supplement 1.
2. Conducted prior to the Product Verification Review.
3. Conducted by individual(s)/organization(s) listed in the IDPD (reference AFLC Supplement 1, paragraph 3-11 to AFR 300-15.)
4. Items to be reviewed made available to auditors by project manager.
5. Results/recommendations given to project manager by auditors.
6. Conducted to make sure the CPCI meets functional requirements and the physical components are correct, complete, and consistent (listings vs logic charts vs narrative definitions, etc.).

N. PRODUCT VERIFICATION REVIEW (PVR)

1. Conducted IAW AFR 300-15, AFLC Supplement 1.
2. Chaired by project manager or his/her designee.
3. Attended by representatives of participating/supporting organizations.
4. Conducted to review all products comprising the CPCI and to make sure all necessary preparations have been made to begin formal environmental testing.

O. CONFIGURATION CONTROL BOARD (CCB) MEETING - See entry II.E.

P. PROGRAM MANAGER REVIEW - See entry II.F.

VII. ACTIONS NECESSARY TO BEGIN THE TEST PHASE

- A. DATA PROJECT DIRECTIVE/UPDATE/AMENDMENT - See entry III.A.**
- B. DATA AUTOMATION REQUIREMENT/AMENDMENT - See entry III.B.**
- C. STATEMENT OF WORK - See entry I.O.**

VIII. ACTIONS NECESSARY TO COMPLETE THE TEST PHASE

A. DATA PROJECT PLAN UPDATE - See entry IV.A.

B. FUNCTIONAL DESCRIPTION CHANGE - See entry IV.B.

C. SYSTEM/SUBSYSTEM SPECIFICATION CHANGE - See entry IV.C.

D. DATA REQUIREMENT DOCUMENT CHANGE - See entry VI.D.

E. DATA BASE SPECIFICATION (DS) CHANGE

1. PMO changes the DS as a result of Baseline Change Requests, which are prompted by DAR(S)/DAR amendments, hardware configuration changes, etc.
2. See entry VI.E. and II.E. for remaining actions/responsibilities.

F. PROGRAM SPECIFICATION (PS) CHANGE

1. PMO changes the PS(s) as a result of Baseline Change Requests, which are prompted by DAR(S)/DAR amendments, changes in functions performed, changes in logic, etc.
2. See entry VI.F. and II.E. for remaining actions/responsibilities.

G. TEST PLAN (PT) CHANGE

1. PMO changes the PT as a result of Baseline Change Requests, which are prompted by DAR(S)/DAR amendments, changes in test plans, conditions, etc.
2. See entry VI.G. and II.E. for remaining actions/responsibilities.

H. MAINTENANCE MANUAL (MM) CHANGE

1. PMO changes the MM as a result of Baseline Change Requests, which are prompted by DAR(S)/DAR amendments, program changes, hardware/software (basic) changes, etc.
2. See entry VI.H. and II.E., for remaining actions/responsibilities.

I. USER'S MANUAL (UM) CHANGES

1. PMO changes the UM IAW AFR 5-1 (the contents of the UM must remain consistent with the other CPCI documentation).
2. See entry VI.I. and II.E. for remaining actions/responsibilities.

J. COMPUTER OPERATION MANUAL (OM) CHANGE

1. PMO changes the OM as a result of Baseline Change Requests, which are prompted by DAR(S)/DAR amendments, procedural changes, program changes, etc.
2. See entry VI.J. and II.E. for remaining actions/responsibilities.

K. TEST ANALYSIS REPORT (RT)

1. Prepared IAW DOD Standard 7935.1-S, AFLC Supplement 1 and AFR 300-15, AFLC Supplement 1.
2. OPR - Project Management Office.
3. OCR - Organizations participating in formal (environmental) testing.
4. Given to audit team prior to final functional/physical configuration audit.
5. Prepared to document the results of testing the ADS functions which are needed to satisfy the "user's" requirements. Also to document known deficiencies and areas of improvement.

L. FINAL FUNCTIONAL/PHYSICAL CONFIGURATION AUDIT (FCA/PCA) - See entry VI.M. for necessary actions/responsibilities.

M. SYSTEM VALIDATION REVIEW (SVR)

1. Conducted IAW AFR 300-15, Chapter 3 and AFLC Supplement 1.
2. Chaired by project manager or his/her designee.
3. Attended by representatives of participating/supporting organizations and "user" representative authorized and responsible for accepting/rejecting developed ADS.
4. Conducted after completion of formal environmental testing for purpose of users certifying that ADS satisfies requirement(s) and is ready for use.

N. CONFIGURATION CONTROL BOARD (CCB) - See entry II.E.

O. PROGRAM MANAGER REVIEW (PMR) - See entry II.F.

IX. OPERATIONAL PHASE ACTIONS

A. DATA PROJECT DIRECTIVE/UPDATE/AMENDMENT - See entry III.A.

B. LETTER OF TRANSMITTAL (TL)

1. Prepared IAW AFR 300-15, Chapter 3 and AFLC Supplement 1 (draft reviewed at preliminary FCA/PCA).
2. OPR - Project Management Office.
3. OCR - Organizations participating in/supporting project.
4. Prepared to send material and instructions needed for ADS testing/implementation.

C. FINAL OPERATIONAL (EVALUATION) REVIEW (FOR)

1. Conducted IAW AFR 300-12, Chapter 4.
2. Chaired by project manager or his/her designee.
3. Attended by representatives of participating/supporting organizations and user representative(s).
4. Conducted after ADS implementation to make sure ADS works satisfactorily after operating for a period of time in an operational environment.

D. PROGRAM MANAGER REVIEW (PMR) - See entry II.F.

I. CONFIGURATION MANAGEMENT (CM)

A. INTRODUCTION

CM is the implementation and maintenance methodology of identification, control, and status accounting, ensuring effective control of the definition/configuration of a desired product throughout the life of the system. The purpose of configuration management is to ensure maintainability of the system throughout its life cycle.

The individual(s) accomplishing the CM functions is responsible for:

Identification and documentation of the functional and physical characteristics of an item.

Controlling changes to those characteristics.

Recording and reporting status of proposed changes.

CM basically deals with the documentation specifying or reporting the status of a configuration item (CI).

The Configuration Management Plan (CMP) is an appendix of the DPP and will be prepared by the project manager. (AFR 300-15, AFLC Sup 1, para 2.2).

The CMP describes responsibilities and procedures for implementing CM within the project and should be completed prior to the SRR. (See AFR 300-15, attach. 4 for format).

The scope of these procedures and number of personnel assigned CM responsibilities should be tailored to the quantity, size, scope, stage of life cycle, nature and complexity of the CI involved, and whether it is government or contractor developed at government expense or privately developed and offered for government use.

A baseline is a specification, under change control, defining a configuration item (CI) at a point in time. These baselines must be maintained for the life of the system.

The functional baseline is defined by the functional description. This is the agreed to definition of the system requirements: performance, operational, logistical, training, etc.

The system/subsystem specifications define the allocated baseline. These are performance-level specifications which define what functions are to be performed by what program(s) or piece of equipment.

The program specifications define the product baseline. These specifications define the components (programs) of the operational system

A computer program is a sequence of coded instructions performing a function or set of functions. It may be a CPCI (below) or part of a CPCI. If a program is developed to assist the development effort, but will not become a component of the operational system, it is not considered to be a CPCI or a part of a CPCI.

A computer program configuration item (CPCI) is the actual coded instructions recorded on a storage medium. It may in fact be a module, program, or a system comprised of programs.

Computer program component (CPC) is a component part of a CPCI. It may be a module or a program.

B. CONFIGURATION IDENTIFICATION

Configuration Identification is the documentation describing the physical and functional characteristics of an item. In the case of DOD computer based systems this documentation is done IAW DODS 7935.1-S. The definition of these characteristics are evolved during the acquisition process and the identification is established upon approval by the customer during the on-going review process.

Designation of a CPCI is done on the basis of the engineering design taking into consideration:

- Project Management Philosophy

- Technical Risk

- Complexity

- Separate Computers

- Separate Schedules

- Different Functions

A CPCI is a level of management, it is the level:

- At which the project manager accepts the product - i.e., program, module, or ADS.

- Below which the development manager accepts all responsibility.

- Above which the project manager is responsible for interface, integration, and performance.

Each CPCI requires:

- Index

- System/Subsystem Specification

- BCR(s) and Associated Status Reports and Files

- Separate Reviews and Audits

- User and Operating Manuals

- Formal Development and Operational Testing

- Formal Acceptance

A Software Documentation Plan (reference AFR 300-15, Chapter 2, para. 2-7 and AFLC Supplement 1.) establishes the documentation to be produced and its management. This includes technical and management documentation, reports and official letters, and standard forms. The following should be covered:

Title and Identification Number

Purpose

Who is Responsible

Cordination and Approval Authorities

Schedule

Publication and Distribution

Documentation standards are covered in DODS 7935.1-S and the AFLC supplement.

C. CONFIGURATION CONTROL

Configuration Control is the practice of maintaining strict change control over requirements, design, code, and test activities. It is the means of ensuring coordination between Project Management, Development Activity Personnel (DAP), and Functional Area Personnel (FAP).

The Configuration Control Board (CCB) is the management activity, chaired by the project manager, which makes all significant decisions relating to baselined documents and baseline change requests.

The Baseline Change Request (BCR) is the recommendation of an alteration to the configuration/definition subsequent to the establishment of the baseline. The proposal includes the statement of the requirement for the proposed alteration as well as the impact on cost, schedule, and performance.

Change Classifications include three types:

A Class I change modifies an established baseline and/or impacts approved cost, schedule or performance. Class I changes must be approved by the CCB prior to implementation.

A Class II change is a documentation change (correction). It does not impact approved cost, schedule, or performance and does not require prior approval by the CCB, only the Project Manager.

Internal changes are those made to a CPCI or document(s) prior to baselining.

NOTE: In the case of a non-government development activity, all Class I and II changes must be concurred with by the Government Procuring Contracting Officer (PCO) prior to implementation.

D. CONFIGURATION STATUS ACCOUNTING

Configuration Status Accounting is the management process of tracking a change from its official recording (BCR, DPR, SPR, DBCR, etc.) until it is disapproved, or approved and officially incorporated into the system and associated documentation. It is also the process of tracking/monitoring the configuration of operational systems.

Records are maintained giving the current position (status) of the item. These record/logs include:

Product Log

Module Log

Software Problem Report Log

Data Base Change Request Log

Baseline Change Request Log

Design Problem Report Log

CPCI Index describes the current status of individual documents. It reports the basic issue or complete revision of each maintainable document and maintains the current status of each with respect to approve BCR(s). It should contain a summary record of dates for development milestones. The BCR log should be used in conjunction with the index and must be consistent. The index is established within 30 days of the allocated baseline.

BCR Log is initiated following the initiation of the first BCR. Its purpose is to portray the status of all BCRs.

E. CONFIGURATION MANAGEMENT FORMS

The Configuration Control Directive (CCD), AFLC Form 406, will document all decisions of the CCB.

The Baseline Change Request, AF Form 1773, formally requests a baseline change. This form, along with the appropriate attachments, comprises the proposal which is considered by the CCB.

The Design Problem Report (DPR), AF Form 1774, is used to document problems identified during reviews and audits.

The Software Problem Report (SPR), AF Form 1775, is used to document a suspected or existing discrepancy or deficiency in a computer program, its documentation, or interfacing hardware.

The Data Base Change Request, AF Form 1776, is used to request a modification to a baselined data base and must accompany an AF Form 1773 if an established baseline is affected. DBCR(s) can be submitted at any point in the life cycle after the document(s) defining contents and/or structure of the data base(s) have been baselined. Some changes may in fact exceed the scope of the DPD and this would require the submission and approval of a DAR/DAR amendment.

The decision of the Project Manager can be to:

Reject the Request

Approve

Approve With Specific Changes (which may be delayed implementation date)

BCR(s) should include:

Identification of Desired Change

Justification

Summary of Alternatives

Identification of Required Tasks, Responsible Agency, and Schedule

Identification of Impacts on Schedule, Cost, and Other Systems/CPCI(s)

The AFLC Form 406 is used for coordination and to reflect approval/disapproval of a Class I change.

The approved modification to the baselined document is done via a change or modification.

A change is a package of pages which have been modified.

A revision is a complete reissue. A revision is required when over 40% of a document has changed or will be changed.

The revision is prepared, issued, and identified the same way the original document was. The revision's specification number will contain a revision letter.

Change pages will contain specification numbers and date.

Specification numbers are assigned by AFLC/ACTM.

Specification numbers must be on each page in the upper corner opposite the binding edge.

Version numbers appear only on the CPCI and associated date/documentation from agency to agency.

F. CM APPLICATION OF CONFIGURATION MANAGEMENT TO ADS DEVELOPMENT

Purpose: to outline method by which HQ AFLC/ACT will implement management audits of selected AFLC LMS development projects.

Configuration management & procedures for application (AFR 300-15, chap. 2).

Reviews and audits in Configuration Management (AFR 300-15, chap. 3).

Specific policy and procedures for application of CM within AFLC (AFR 300-15, AFLC Supp 1).

Authority for AFLC/ACT to conduct audits (AFR 300-15, AFLC Supp 1, para 2-22b; AFLCR 400-18, para 2-10g) including review of DPDs and DPPs.

Reference: AFR 300-15, AFLC/AC OI 171-6.

III. QUALITY ASSURANCE

Quality Assurance (QA) is the implementation of a series of planned activities whose objective is to provide confidence that the end products conform to established technical requirements and performance standards.

A QA Plan is produced based on a review of the DAR, DPD, DPP, and FD. The plan describes QA functions, responsibilities, and tools (reference AFR 300-15, Chapter 4 and AFLC Supplement 1).

The QA Program impacts Configuration Management, Testing, Data Management, Technical Standards, Program Design and Coding Standards, Reviews and Audits.

Software QA must be life-cycle-oriented to reduce maintenance costs.

There is not an AFLC software quality assurance program. Therefore it is the responsibility of the project manager to ensure a "Quality Product" is delivered.

IV. MCAP

HQ USAF/ACD requires a new MCAP each year by 1 Oct., six copies (AFR 300-7, para 8d(1)).

The plan covers the four year period beginning two years after date submitted (AFR 300-7, para 8d(1)).

Its purpose is to document ADP requirements as an aid to planning, budgeting and management (AFR 300-7, para 8a,b).

AFLC Management Systems Panel will review and approve the MCAP subject to review by AFLC Planning and Programming Review Board (AFR 300-7, AFLC Sup 1, para 3e and 8c(4)(c)).

HQ AFLC/ACDR is the focal point for MCAP policies and procedures. They prepare the draft plan and distribute the approved plan.

HQ AFLC/ACD will prepare MCAP Sect. 1 Command Situation (AFR 300-7, AFLC Sup. 1, para 8c(4)(b) and forward to arrive in ACDR not later than 15 July (AFR 300-7, AFLC Sup 1, para 8d(5)).

HQ AFLC/ACDS will prepare MCAP Sect. 2 ADPS Plan Summaries (AFR 300-7, AFLC Sup 1, para 8c(4)(b)(2)) and forward to arrive in ACDR not later than 15 July (AFR 300-7, AFLC Sup 1, para 8d(5)).

HQ AFLC/ACD will prepare Sect. 3 and 4, Projected Automation Requirements (AFR 300-7, AFLC Sup 1, para 8c(4)(b)3a thru 3d).

Preparation of PARs is continuing function. PARs received by 15 July will be included in the MCAP for the same calendar year (AFR 300-7, AFLC Sup 1, para 8d(4)).

Corrections to MCAP required by ACD will be effected within 21 days from receipt of notice. Other changes or updates will be submitted 21 days before effective date (AFR 300-7, para 8d(2)). These changes will be approved by AFLC Management Systems Panel; sufficient leadtime must be allowed (AFR 300-7, AFLC Sup 1, para 8d(2)).

V. EA/MILESTONE REPORT

A. AF FORM 2053

Economic Analysis (Summary of Alternatives)

Purpose: to compare the net costs for each alternative, excluding baseline and augmented current system.

Get discount and uniform annual rates from AFR 178-1.

Get Net Annual Costs from AF Form 2054.

Reference: AFR 300-12, para 3-10a, 3-11.

Example: AFR 300-12, Attachment 7.

B. AF FORM 2054

Economic Analysis (Summary of Alternative No. _____)

Purpose: to compute net cost of an alternative, even the baseline and augmented system.

Obtain the data for:

<u>Part</u>	<u>Form</u>
I	AF Form 2055
IID.	AF Form 2056
IV	AF Form 2057

Aggregate parts I and II to form part III.

Combine parts III and IV to form part V.

Part V becomes the input to AF Form 2053.

References: AFR 300-12, para 3-10b, 3-12.

Example: AFR 300-12, Attachment 8.

C. AF FORM 2055

Economic Analysis (ADPMIS Costs)

Purpose: to compute the detailed ADPMIS Costs of the baseline and each alternative.

Use a separate form for each phase of the life cycle, (AFR 300-12, para 3-3).

Use a separate form for each DPI, Program Element Code (PEC), Command, or Appropriation if more than one are reportable.

Get cost categories from RCS: DD-COMP(AR)996.

Get salary factor guidance from ACD and include the schedule of factors identified by source and date with each analysis. If using average rates for personnel significantly affects choice between alternatives, (AFR-300-12, para 3-13d).

Line H, the "net" ADPMIS COSTS, becomes the input to Section I, AF Form 2054.

References: AFR 300-12, para 3-106, 3-13.

Examples: AFR 300-12, Attachment 9.

D. AF FORM 2056

Economic Analysis (Non-MIS Costs)

Non-MIS costs include TDY, training, and others not directly identified with ADP but which are directly related to new systems development.

Report costs in thousands of dollars.

Use a separate form for each phase of the life cycle.

Use a separate form for each DPI, Program Element Code, Command, or Appropriation if more than one are reportable.

Cross reference costs elements with the milestone reporting system, AF Form 2060.

Get salary factor guidance from ACD and include the schedule of factors identified by source and date. If using average rates for personnel significantly affects the choice between alternatives (AFR 300-12, para 3-13d).

Give unique costs on line D and explain in a footnote or addendum.

Line E, the total Non-MIS costs, becomes the input to Section IV, AF Form 2054.

Reference: AFR 300-12, para 3-10c, 3-14.

Example: AFR 300-12, Attachment 10.

E. AF FORM 2057

Economic Analysis (Cost-Reduction Savings)

Purpose: to report the savings resulting from implementing an alternative.

See AFR 300-12, para 3-9(2) for discussion of costs and savings.

Use a separate form for each DPI, Program Element Code (PEC), Command, or Appropriation if more than one are reportable.

Get data for Part I from AF Form 2055 for baseline system.

Calculate Part II, Augmentation Avoided. Get data from AF Form 2055 for augmented system and subtract cost of baseline system obtained above.

Include in Part III any other savings that can be identified to specific budgets or major programs.

Lines I, II, and III B3 become inputs to Section IV, AF Form 2054.

References: AFR 300-12, para 3-10c, 3-15.

Example: AFR 300-12, Attachment 11.

F. AF FORM 2058

Milestone Reporting System (Schedule)

Purpose: to describe the milestones for an ADPS/ADS project.

All dates are given in six digit form (YYMMDD).

Key milestones and their codes are given in AFR 300-12, para 3-21d(1).

Include other reviews/audits specified in DPD, e.g., SDR, PDR, CDR. Do not use codes except as above.

A short title may be given under "Remarks"

Note the identification of the rows on this form with the columns on AF Forms 2059 and 2060.

References: AFR 300-12, para 3-18, 3-21.

Example: AFR 300-12, Attachment 12.

G. AF FORM 2059

Milestone Reporting System (ADPMIS Costs)

Purpose: to compute the variance analysis by cost element for ADPMIS costs.

Note the cross reference of cost categories with AF Form 2055.

Each report must show costs for the current milestone, the next three milestones and the fully operational milestone.

Columns D and H record cumulative time and costs between milestones. Column D gives actual cost to date; column H projects total cost.

Specific details for each column are given in AFR-300-12, para 3-22.

Report is due not later than 30 days after the scheduled milestone date. If actual completion date will be later than scheduled date, notify HQ USAF/ACDC of the reason and the anticipated submission date.

Management reviews are required if the incremental cost exceeds the approved cost by 25% or the milestone slips 120 days. (AFR-300-12, para 3-23).

H. AF FORM 2060

Milestone Reporting System (Non-MIS Costs)

Purpose: to compute the variance analysis by cost element for Non-MIS costs.

Note the cross reference of cost categories with AF Form 2056.

Each report must show costs for the current milestone, the next three milestones and the fully operational milestone.

Columns D and H record cumulative time and costs. All other columns reflect incremental costs between milestones. Column D gives actual cost to date; column H projects total cost.

Specific details for each column are given in AFR-300-12, para 3-22.

Report is due not later than 30 days after the scheduled milestone date. If actual completion date will be later than scheduled date notify HQ USAF/ACDC of the reason and the anticipated submission date.

Management reviews are required if the incremental cost exceeds the approved cost by 25% or the milestone slips 120 days. (AFR-300-12, para 3-23).

VI. COMMUNICATIONS CONSIDERATION

Telecomm memo entry is made to PAR and the Telecommunications Plan (C3P2).

During DAR preparation Comm Requirement is determined and defined to AFLC/DCO IAW AFR 300-12, AFLC Supp 1, Atch 6, para 30. (OPR-Project Manager)

Communications alternatives examined for cost (OPR: AFLC/DCO)

A communications concept is selected and discussed with Project Manager (AFLC/DC)

A Communication Annex is prepared IAW AFR 100-18 for inclusion in the DAR or DPP (OPR: AFLC/DCO)

APPENDIX G

GLOSSARY OF TERMS FOR SOFTWARE CONFIGURATION MANAGEMENT

(Ref 141)

GLOSSARY OF TERMS FOR CONFIGURATION MANAGEMENT*

Acceptance

An official act by the customer to accept transfer of accountability, title, and delivery of a Configuration Item or other items on a Contract (e.g., Data Item).

Acceptance Testing - Subset of Qualification Tests

The formal inspection, testing, and/or analysis accomplished in accordance with Section 4 of a Product Specification to verify the performance and adequacy of a production item.

Approved Change

A change for which approval has been given for change incorporation by the applicable change control board.

Audits

Configuration audits verify conformance to specifications and other contract requirements. Audits are not reviews.

NOTE: Audits differ from reviews in that reviews are conducted on a periodic basis to assess the degree of completion of technical efforts related to identified milestones before proceeding with further technical effort.

- a. Functional Configuration Audit (FCA). The formal examination of functional characteristics' test data for a configuration item, prior to acceptance, to verify that the item has achieved the performance specified in its functional or allocated configuration identification.
- b. Physical Configuration Audit (PCA). The formal examination of the "as-built" configuration of a unit of a CI against its technical documentation in order to establish the CI's initial product configuration identification.

*This list is by no means complete but contains those terms most frequently used in the industry. The list has been compiled from a variety of industry and government sources.

Baseline

A configuration identification document or a set of such documents formally designated and fixed at a specific time during a CI's life cycle. Baselines, plus approved changes from those baselines, constitute the current configuration identification. For configuration management, there are three baselines, as follows:

- a. **Functional Baseline.** The initial approved functional configuration identification.
- b. **Allocated Baseline.** The initial approved allocated configuration identification.
- c. **Product Baseline.** The initial approved or conditionally approved product configuration identification.

Computer Equipment

Devices capable of accepting and storing computer data, executing a systematic sequence of operations on computer data or producing control outputs. Such devices can perform substantial interpretation, computation, communication, control, and other logical functions.

Computer Firmware

A microcoded control program that resides in a Programmable Read Only Memory (PROM) or Read Only Memory (ROM).

Computer Program

A series of instructions or statements in a form acceptable to computer equipment, designed to cause the execution of an operation or series of operations. Computer programs include such items as operating systems, assemblers, compilers, interpreters, data management system, utility programs, and maintenance/diagnostic programs. They also include applicable programs such as payroll, inventory control, operational flight, strategic, tactical, automatic test, crew simulator, and engineering analysis programs. Computer programs may be either machine dependent or machine independent, and may be general purpose in nature or be designed to satisfy the requirements of a specialized process of a particular user.

Computer Program Component (CPC)	A CPC is a functionally or logically distinct part of a computer program configuration item (CPCI), distinguished for purposes of convenience in designing and specifying a complex CPCI as an assembly of subordinate elements.
Computer Program Configuration Item (CPCI)	An aggregate of computer programs, or of the discrete portions (e.g., CPCs, modules, routines, etc.) thereof, which satisfy an end use function and which is designated by the Government for configuration management. CPCIs may vary widely in complexity, size, and type, from a special purpose diagnostic program to a large command and control system.
Computer Program Library	A controlled collection of program source statements on media such as punched cards, magnetic tapes, or discs. This library is also the repository for the controlled master copies of each computer program.
Computer Resources	The totality of computer equipment, computer program, computer data, associated documentation, personnel, and supplies.
Configuration	The functional and/or physical characteristics of hardware/software as set forth in technical documentation and achieved in a product.
Configuration Control	The systematic evaluation, coordination, approval or disapproval, and implementation of all approved changes in the configuration of a CI/CPCI after formal establishment of its configuration identification.
Configuration Control Board	A board composed of representatives from program/project functional areas such as engineering, configuration management, procurement, production, test and logistic support, training activities, and using/supporting organizations. This board approves or disapproves proposed engineering changes with each member recording his organization's official position. The program/project manager is normally the board chairman and makes the final decision on all changes unless otherwise directed by command policy. The board issues a directive/request to implement its decision.

Configuration Identification

The current approved or conditionally approved technical documentation for a configuration item as set forth in specifications, drawings, and associated lists, and documents referenced therein.

Configuration Item (CI)

An aggregation of hardware/software, or any of its discrete portions, which satisfies an end use function and is designated by the Government for configuration management. CIs may vary widely in complexity, size, and type, from an aircraft, electronic or ship system to a test meter or round of ammunition. During development and initial production, CIs are only those specification items that are referenced directly in a contract (or an equivalent in-house agreement). During the operation and maintenance period, any reparable item designated for separate procurement is a configuration item.

Configuration Management

A discipline applying technical and administrative direction and surveillance to (1) identify and document the functional and physical characteristics of a configuration item, (2) control changes to those characteristics, and (3) record and report change processing and implementation status.

Configuration Status Accounting

The recording and reporting of the information that is needed to manage configuration effectively, including a listing of the approved configuration identification, the status of proposed changes to configuration, and the implementation status of approved changes.

Contract

The legal agreement between DOD and industry, or similar internal agreement wholly within the government, for the development, production, maintenance, or modification of an item(s).

**Contract Data Requirements
LIST (CDRL)**

A contract form, DD 1423, listing all technical data items, selected from an Authorized Data List, to be delivered under the contract.

Contractor

An individual, partnership, company, corporation, or association having a contract with the procuring activity for the design, development, design and manufacture, manufacture, maintenance, modification or supply of items under the terms of a contract. A government activity performing any or all of the above actions is considered to be a contractor for configuration management purposes.

Cost

The term "cost" means cost to the Government.

Non-Recurring Costs

One-time costs which will be incurred if an engineering change is ordered and which are independent of the quantity of items changed, such as: cost of redesign, special tooling, or qualification.

Recurring Costs

Cost which are incurred for each item changed or for each service or documented ordered.

Critical Item

An item within a configuration item (CI) which, because of special engineering or logistic considerations, requires an approved specification to establish technical or inventory control at the component level.

Critical Design Review (CDR)

This review shall be conducted by the developer for each configuration item when detail design is essentially complete. The purpose of this review will be to determine that the detail design of the configuration item under review satisfies the design requirements established in the configuration item specification, and establishes the exact interface relationships between the configuration item and other items of equipment and facilities.

Data (Technical Data and Information)	The means for communication of concepts, plans, descriptions, requirements, and instructions relating to technical projects, material, systems, and services. These may include: specifications, standards, engineering drawings, associated lists, manuals, and scientific and technical reports. They may be in the form of documents, displays and sound records, punched cards, and digital or analog data.
Data Package	A collection of data products (items) which is complete for a specific use.
Decommissioning	Change in program status from operational to inactive.
Debugging	A process to detect and remedy inadequacies, preferably prior to formal testing and operational use.
Deficiencies	Deficiencies consist of two types: conditions or characteristics in hardware/software which are not in compliance with a specified configuration, or inadequate (or erroneous) configuration identification which has resulted or may result, in configuration items that do not fulfill approved operational requirements.
DOD Components	Term which describes the Office of the Secretary of Defense, the Military Departments, Office of the Joint Chiefs of Staff, and the Defense Agencies.
Documentation	The specifications, reports, plans and procedures used to document and support computer programs.
DSARC	The Defense Systems Acquisition Review Council; an advisory body to the Secretary of Defense on major system acquisition programs and related policies.
Embedded	Adjective modifier; integral to, from the design, procurement, and operations point of view, espoused in DOD Directive 5000.1. For computer programs, this implies those programs which are integral to, but part of a larger system.

Engineering Change

An alteration in the configuration of a configuration item or an item, delivered or to be delivered, or under development, after formal establishment of its configuration identification. Changes may be Class I or II.

Engineering Change Priorities

The rank assigned to a Class I engineering change, which determines the methods and resources to be used in review, approval, and implementation. There are three recognized priorities:

- Emergency
- Urgent
- Routine

Engineering Change Proposal

A term which includes both a proposed engineering change and the documentation by which the change is described and suggested.

ECP Types

A term covering the subdivision of ECPs on the basis of the completeness of the available information delineating and defining the engineering change, i.e., preliminary or formal ECP.

Form, Fit, and Function

That configuration comprising the physical and functional characteristics of the item as entity but not including any characteristics of the elements making up the item.

Formal Qualification Review (FQR)

The FQR is a review to ensure that the quality assurance tests have been accomplished that verify that the item as designed performs as required by the specification performance requirements. An FQR is held for each new design Computer Program Configuration Item. (MIL-STD-483)

Function

A discrete action required to achieve a given objective, to be accomplished by hardware, computer program, personnel, facilities, procedural data, or a combination thereof. It is an operation the system must perform in order to fulfill its intended mission.

Functional Area

A distinct group of system performance requirements which, together with all other such groupings, form the next lower level breakdown of the system on the basis of function.

Functional Characteristics

Quantitative performance, operating and logistic parameters and their respective tolerances. Functional characteristics include all performance parameters, such as range, speed, lethality, reliability, maintainability, safety.

Functional Configuration Identification (FCI)

The current approved technical documentation for a configuration item which prescribes: (1) all necessary functional characteristics, (2) the tests required to demonstrate achievement of specified functional characteristics, (3) the necessary interface characteristics with associated CIs, (4) and CIs key functional characteristics and its key lower level CIs, if any, and (5) design constraints, such as envelope dimensions, component standardization, use of inventory items, integrated logistics support policies.

Hardware/Software

Hardware or software, or a combination of both, in which the software includes only that associated with hardware for operational use, e.g., computer programs for command and control, handbooks for operations, maintenance, etc., and excludes fabrication specifications, drawings, etc.

Integrated Logistic Support

A composite of the elements necessary to assure the effective and economical support of a system or equipment at all levels of maintenance for its programming life cycle. The elements include all resources necessary to maintain and operate an equipment or weapons system, and are categorized as follows: (1) planned maintenance, (2) logistic support personnel, (3) technical logistic data and information, (4) support equipment, (5) spares and repair parts, (6) facilities, and (7) contract maintenance.

Item (When the term is used without a modifier)	Any level of hardware assembly below a system; i.e., subsystem, equipment, component, subassembly, or part. Also see "Configuration Item", "Critical Item", and Privately Developed Item".
Interface Control Working Group (ICWG)	A group chartered as the official channel among contractors, the customer, and other agencies to resolve interface problems, exchange new interface information, document change agreements, and coordinate Engineering Change Proposals affecting interfaces within a project.
Master	The official version of a document, system, or program.
Maintenance/Modification to a Program	The maintenance of a computer program is defined to be any modification to the instruction of an operating digital computer program or system for corrective or improvement purposes.
Mnemonic	A brief alphabetical title used to refer to a program, subprogram, subroutine, etc. It often gives information about the program function or serves as a memory aid in this regard.
Object Form	Coded instructions that can be acted upon directly by a computing system without requiring an additional processing step.
Operational Systems Development	Includes a research and development effort directed toward development, engineering, and test of systems, support programs, vehicles, and weapons that have been approved for production and Service Employment.
Physical Characteristics	Quantitative and qualitative expressions of material features, such as composition, dimensions, finishes, form, fit and their respective tolerances.

Preliminary Design Review (PDR)

This review shall be conducted for each configuration item prior to the detail design process to: evaluate the progress and technical adequacy of the selected design approach; determine its compatibility with the performance requirements of the configuration item Development Specification; and establish the existence and compatibility of the physical and functional interfaces between the configuration item and other items of equipment of facilities.
(MIL-STD-1521)

Product

A software product is a computer program and its associated documentation. It may be in the form of card decks, magnetic tapes, disk-packs, or other physical media capable of transmitting its contents directly to a computer.

Privately Developed Item

An item completely developed at private expense and offered to the Government as a production article, with Government control of the article's configuration normally limited to its form, fit, and function.

Project Work Breakdown Structure (Project WBS)

A project WBS is defined as the complete WBS for the project, containing all WBS elements related to the development and/or production of the defense material item. The project WBS evolves from the project summary WBS extended to include all contract WBS(s) and equivalent WBS(s) resulting from DOD in-house efforts. The WBS is used for cost accounting and monitoring of the project.

Release

The transfer of physical custody and control of products or documentation from the originator to another organization in a controlled environment, such as through a formal release system or organization.

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/O 9/2
SOFTWARE QUALITY METRICS: A SOFTWARE MANAGEMENT MONITORING METH--ETC(U)
MAR 82 S J JARZOMBK
AFIT/SCS/NA/82M-1
NL

ML

Δ_{1F4}
Δ_{1F5} 50

END
DATE
FILMED
7-8
DTIC

Retrofit (Retroactive Retrofit)	Modification of a configuration item to incorporate changes made in later production of a similar type.
Software	A combination of associated computer programs and computer data required to enable the computer equipment to perform computational or control functions.
Software Engineering	Science of design, development, implementation, test, evaluation, and maintenance of computer software over its life cycle.
Software Data Package	A compilation or physical grouping of all required documentation: listings, card decks, tapes, etc., to support a computer program.
Source Form	Coded instructions written in one of several programming languages that cannot be directly acted upon by a computing system without requiring an interim processing conversion to a machine language (also see 'Object Form').
Software Reliability	The probability that a computer program configuration item will perform its intended function for a specified interval under stated conditions.
Specification	A document intended primarily for use in procurement, which clearly and accurately describes the essential technical requirements for items, materials or services including the procedures by which it will be determined that the requirements have been met.
Specification Addendum	This data item is used to: create a new computer program configuration item which is similar to an existing computer program configuration item, with minimum redesign effort, and apply a formal means of writing a specification for a new computer program configuration item by changing an existing specification for a computer program configuration item in a manner which permits ready comparison of the exact relationship between two computer program configuration items.

**Specification Change
Notice (SCN)**

This document is used to propose, transmit, and record changes to a specification. In the proposal or preliminary form, prior to approval, the SCN supplies copies of the pages containing the proposed changes.

System Specification

A document which states the technical and mission requirements for a system as an entity, allocates requirements to functional areas (or configuration items), and defines the interfaces between or among the functional areas.

Development Specification

A document applicable to an item below the system level which states performance, interface, and other technical requirements in sufficient detail to permit design, engineering for service use, and evaluation.

Product Specification

A document applicable to a production item below the system level which states item characteristics in a manner suitable for procurement, production, and acceptance.

Function (Performance)

A product specification which states: (1) the complete performance requirements of the product for the intended use, and (2) the necessary interface and interchangeability characteristics. It covers form, fit, and function.

Fabrication (Design)

A product specification which states: (1) a detailed description of the parts and assemblies of the product, usually by prescribing compliance with a set of drawings, and (2) those performance requirements and corresponding tests and inspections necessary to assure proper fabrication, adjustment, and assembly techniques.

System Design Review (SDR)

This review shall be conducted when the definition effort has proceeded to the point where system requirements and the design approach are more precisely defined, (i.e., alternate design approaches and corresponding test requirements have been considered and the contractor has defined and selected the required equipment, logistic support, personnel, procedural data, and facilities). This review shall be in sufficient detail to ensure a technical understanding between the contractor and the procuring activity on: the system segments identified in the system specification and the configuration items identified in the configuration item performance specification(s). (MIL-STD-1521)

System Requirements Review (SRR)

The objective of this review is to ascertain the adequacy of the contractor's efforts in defining system requirements. It will be conducted when a significant portion of the system functional requirements has been established. (MIL-STD-1521)

Subcontractor

A "subcontractor" is an individual, partnership, corporation, or association, who (which) contracts with a contractor to design, develop, design and manufacture, manufacture items, which are or were, designed specifically for use in military application.

System

A composite of subsystems, assemblies (or sets), skills, and techniques capable of performing and/or supporting an operational (or non-operational) role. A complete system includes related facilities, items, material, services, and personnel required for its operation to the degree that it can be considered a self-sufficient item in its intended operational (or non-operational) and/or support environment.

System/Subsystem Specification

This specification may be prepared to guide development of large projects and may be used to prepare individual subsystem specifications. The system/subsystem is a technical document prepared to describe the system. It is detailed as much as possible concerning environment and design to provide maximum guidance for program design. All system/subsystem interfaces are defined. (SECNAVINST 5233.1)

Test Plan

The acceptance test plan, provides an overall integrated outline of the total test program, program, including: test objectives, identification of test areas, and responsibilities.

Test Report

A document containing the results and analyses of tests executed during validation and acceptance testing.

Vendor

A "vendor" is a manufacturer or supplier of a commercial item.

Version Description Document

This data item shall be used to accompany changes to an approved and released computer program. Its purpose is to identify the changes made and the exact version of a computer program to be delivered.

Waiver

A written authorization to accept a configuration item or other designated items, which during production or after having been submitted for inspection, are found to depart from specified requirements, but nevertheless are considered suitable for use "as is" or after rework by an approved method.

APPENDIX H
GLOSSARY FOR SQA TOOLS AND TECHNIQUES

TECHNIQUES

The Twenty-One Techniques are Listed
in Tables X, XI, and XII

1. **ALGORITHM EVALUATION TEST.** A technique used to evaluate critical algorithm trade-offs (i.e., speed versus size versus precision) before the design is finalized. Often called "the hardest out first method," the technique creates a detailed design based upon trial coding results for key algorithms. The algorithms are often extensively exercised in a simulated environment to ensure mission requirements are satisfied.

2. **ANALYTICAL MODELING.** A technique used to express mathematically (usually by a set of equations) a representation of some real problem. Such models are valuable for abstracting the essence of the subject of inquiry. Because equations describing complex systems tend to be complicated and often impossible to formulate, it is often necessary to make simplifying assumptions, which may tend to distort accuracy (Ref 90).

3. **AUDITING.** A formal technique employed to examine and verify through inspection either the status of a program and its documentation or the adherence of project personnel to established procedures. Scheduled audits are normally contractually imposed and periodically held. Unscheduled audits are utilized at random intervals to assess compliance with quality requirements.

4. **CODE INSPECTION.** A disciplined technique used for inspecting the code and identifying errors. Participants have well-defined roles and criteria for evaluating the code. If errors are identified, the code is reworked. Follow-up procedures are used to ensure that the errors have been corrected (Ref 29).

5. **CORRECTNESS PROOFS.** A technique used to prove the correctness of programs using means similar to those employed to prove mathematical theorems. Axioms and theorems derived are used to establish the validity of the program with respect to a precise specification of its purpose. The most frequently used method is known as the inductive assertion or Floyd method (Ref 91). Several approaches are being pursued. One approach seeks to demonstrate program correctness a priori by establishing the proof prior to implementation. Another approach uses

an interactive system to prove correctness a posteriori (Ref 92).

6. **DESIGN INSPECTION.** A disciplined technique used for inspecting the design and identifying errors. Participants have well-defined roles and criteria for evaluating the design. If errors are identified, the design is reworked. Follow-up procedures are used to ensure that the errors have been corrected (Ref 29).

7. **ERROR-PRONE ANALYSIS.** A technique employed during coding to identify areas of the program that have required abnormally frequent correction and change. These areas can either be reworked or subjected to an extensive test effort (Ref 93).

8. **EQUIVALENCE CLASSES.** A technique used to automatically identify a complete set of test cases for a program. The set is interpreted in terms of inequalities involving program variables that define a set of conditions necessary for the particular program flow to actually occur. Some experience in the practical application of the technique has been reported (Refs 94, 95).

9. **EXECUTION ANALYSIS.** A technique employed during test to investigate program behavior errors and to identify areas in the code that were either untested or not fully tested. The program is executed and statistics are collected. Test results and the statistics are then analyzed to insure that each interface, functional and test requirement has been correctly mechanized by the code.

10. **FUNCTIONAL TESTING.** A technique used to demonstrate that the software performs its specifications satisfactorily under normal operating conditions, computing nominally correct output values from nominal input values (Ref 96).

11. **LOGICAL TESTING.** A technique used to confirm that the code performs its computation correctly. Items validated by logical testing include arithmetic (i.e., precision, accuracy, etc.), error handling, initialization, interfaces, and timing (Ref 96).

12. **PATH TESTING.** A technique used to confirm that certain test-effectiveness measures based on the program's control topology have been realized. The technique assures that a sufficient number of statements, branch paths, and subroutine calls have been exercised during the program execution. It also helps identify a complete set of test cases for the program (Ref 97).

13. **POST-FUNCTIONAL ANALYSIS.** A technique employed after completion of functional testing to identify functionally weak areas in the program. The recorded test results are

analyzed and the quality of the final product is determined (Ref 93).

14. REVIEWING. A technique employed to examine and verify through inspection either the status of a program and its documentation or adherence of project personnel to established procedures. Scheduled reviews are normally contractually imposed and periodically held (Ref 87). Informal reviews are held frequently to assess in detail the technical adequacy of the software product (Ref 98).

15. SIMULATION. Simulation is the process of studying specific system characteristics by use of models exercised over a period of time and a variety of conditions for the purpose of evaluating alternatives, timing, system capacities, performance, and constraints within the confines of that system (Ref 99). Simulation can be used by quality assurance throughout the life cycle. It can assist in evaluating conceptual trade-offs (Ref 100). It can also be used to model the environment and provide realistic test inputs to a program being examined.

16. SOFTWARE QUALITY METRICS. A technique used to quantitatively assess the quality of software throughout the SDLC. Measurements or metrics provide a software monitoring method which gives an indication of the progression toward a specified quality (Ref 17).

17. STANDARDIZATION. A technique used to create an authoritative model against which products and/or procedures can be compared in order to determine their quality. Software items for which standards can be easily established include documentation (Refs 101, 102), languages (Refs 103, 104), designs (Refs 105, 8, 106), and structured programming (Ref 107).

18. STATIC ANALYSIS. A technique employed during test to identify weaknesses in the source code. The syntax of a program is examined and statistics about it are generated. Items such as relationships between module, program structure, error-prone constructions, and symbol/subroutine cross-references are checked and violations of established rules are analyzed.

19. STRESS TESTING. A technique employed to confirm that the code performs its specifications satisfactorily under extreme operating conditions, computing nominally correct output values from worst case input values (i.e., singularities, end points for the range of data, etc.).

20. SYMBOLIC EXECUTION. A technique that employs symbolic data to confirm that the software performs properly. Symbolic execution allows one to choose intermediate points in the test spectrum ranging from individual test runs to correctness proofs. Its results can be used to develop a

minimum set of test cases (Refs 108, 109).

21. WALK-THROUGHS. A technique used for reviewing the design or code and identifying errors. The responsible programmer discusses his product with his peers and solicits their constructive advice. Product modifications are then made at the discretion of the programmer to correct problems identified during the review (Ref 110).

TOOLS

The Thirty-Three Tools are Listed
in Tables X, XIII, and XIV

1. **ACCURACY STUDY PROCESSOR.** A computer program used to perform calculations or assist in determining if program variables are computed with required accuracy. The processor accepts mathematical equations and data as inputs. It then uses the data as variables in the equations and solves them (Ref 111).

2. **AUTOMATED MEASUREMENT TOOL.** A computer software package that collects development data through manual and automatic input. The AMT derives metric scores which reflect the level of quality that the developing software exhibits. It generates reports that show a quantitative measure of the quality attributes and factors being considered (Refs 17, 23, 88).

3. **AUTOMATED TEST GENERATOR.** A computer program that accepts inputs specifying a test scenario in some special language, generates the exact computer inputs, and determines the expected results. The NASA-developed Automated Test Data Generator (ATDG) serves as one example (Ref 112). ATDG takes identified code segments, determines all possible logic transfers between those segments, defines a path through the module under test, and generates input data required to execute the selected paths. The General Electric automated software test driver serves as another example (Ref 113). This program automates the process of test planning, test setup, test execution, and test analysis. Other generators are described in a recent report (Ref 114).

4. **COMPARATOR.** A computer program used to compare two versions of the same computer program under test to establish identical configuration or to specifically identify changes in the source coding between the two versions (Ref 90).

5. **CONSISTENCY CHECKER.** A computer program used to determine (1) if requirements and/or designs specified for computer programs are consistent with each other and their data base and (2) if they are complete. The consistency checker computer program developed by TRW is an example (Ref 115).

6. CROSS-REFERENCE. A group of computer programs that provide cross-reference information on system components. For example, programs can be cross-referenced with other programs, macros, parameter names, etc. This capability is useful in problem-solving and testing to assess impact of changes to one area or another (Ref 90). This capability should be provided in most compiler environments (Ref 116).

7. DATA BASE ANALYZER. A computer program that reports information on every usage of data, identifies each program using any data elements, and indicates whether the program inputs, uses, modifies, or outputs the data element. Any unused data is printed. Errors dealing with misuse and nonuse of data and conflicts in data usage are identified during the analysis (Ref 90).

8. DEBUGGER. Compile and execution-time check-out and debug capabilities that help identify and isolate program errors. They usually include commands or directives such as DUMP, TRACE, MODIFY CONTENTS, BREAKPOINT, etc. (Ref 90). Some debuggers operate at the source level (e.g., SIMDDT for SIMULA 67, and COBDDT for COBOL on the PDP-10) and others at the objective level with some additional source information (e.g., DDT for FORTRAN on the PDP-10) (Ref 117). The following types of basic information are normally provided:

TRACE: A timed record of program execution and machine environment information.

DUMP: A record of selected portions of memory or register usage output after a specified point in the program's execution has been reached.

SNAP: A record of intermediate values of items captured during program execution.

BREAKPOINTS: A facility whereby the normal computation is interrupted and debugging activities commence execution.

9. DECISION TABLES. A mechanism used to represent information on program conditions, rules, and actions in tabular form that can be automatically translated to executable code by a processor. Decision tables are a tabular representation of the design which can be used to clarify the control flow of decision alternatives by presenting the information in a concise and understandable format (Ref 99). A dated but useful survey of automated decision table processors is available (Ref 118).

10. DYNAMIC ANALYZER. A computer program that instruments the source code by adding counters and other

statistics-gathering sensors and produces reports on how thoroughly the various portions of the code have been exercised after the augmented code is executed. Dynamic analyzers provide information useful for tuning, optimization, and test case design (Ref 90). Many dynamic analyzers have been developed. Comparative analysis of several analyzers (Ref 119) and user feedback with such systems have been published (Ref 120).

11. DYNAMIC SIMULATOR. A computer program used to check out a program in a simulated environment comparable to that in which it will reside. Closed-loop effects between computer and environmental models are gained when the various models respond to inputs and outputs. The simulator allows the environment to be stabilized at a specific configuration for any number of runs required to observe, diagnose, and resolve problems in the operational program (Ref 90). The Dynamic Test Station (DTS) being developed by the U.S. Air Force to support the test of the operational flight software for the F-16 fighter aircraft serves as an example (Ref 121).

12. EDITOR. A computer program used to analyze source programs for coding errors and to extract information that can be used for checking relationships between sections of code. The editor can scan source code and detect violations to specific programming practices and standards, construct an extensive cross-reference list of all labels, variables and constants, and check for prescribed program formats (Ref 90).

13. FLOWCHARTER. A computer program used to show in detail the logical structure of a computer program. The flow is determined from the actual operations as specified by the executable instructions, not from comments. The flowcharts generated can be compared to flowcharts provided in the computer program design specification to show discrepancies and illuminate differences (Ref 90). Several flowcharters such as AUTOFLOW and FLOWGEN are commercially available. Several structured flowchart representations have been proposed (Ref 122).

14. HARDWARE MONITOR. A unit that obtains signals from a host computer through probes attached directly to the computer's circuitry. The signals obtained are fed to counters and timers and are recorded. These data are then reduced to provide information about system and/or program performance (CPU activity, channel utilization, etc.) (Ref 123). Several useful articles on using this tool have appeared in the literature (Refs 124, 125).

15. INSTRUCTION TRACE. A computer program used to record every instance a certain class of operations occurs and triggers event-driven data collection. In some cases, this creates a complete timed record of events occurring during

program execution (Ref 90). Experience using traces to locate sources of nonrepeatable, intermittent malfunctions has been reported (Ref 126).

16. INTERFACE CHECKER. A computer program used to automatically check the range and limits of variables as well as the scaling of the source program to assure compliance with interface control documents. Other computer programs have been developed to automatically verify that modularity rules have been followed (Ref 127).

17. INTERRUPT ANALYZER. A computer program that determines potential conflicts to a system as a result of the occurrence of an interrupt (Ref 90).

18. LANGUAGE PROCESSORS. Computer programs used to translate high-level or symbolic instruction mnemonics into computer-oriented code capable of being executed by a computer. Compilers, assemblers and meta-assemblers are example tools used for program development. Other language processors have been developed to support requirements generation and validation (Ref 128), design (Ref 129), and test (Ref 113). Preprocessors have been developed to support implementation of modern programming techniques.

19. LIBRARIES. A collection of organized information used for reference or study. Many varieties of library systems can be implemented. Some manage the storage and distribution of the computer program in both source and object form. Others manage the computer program, its documentation and related test data (i.e., test cases, procedures, results). Programs which support their implementation are commercially available and include Applied Data Research's LIBRARIAN and International Business Machine's Program Production Library (PPL) (Ref 90). The use of a library and its effect on productivity have been reported (Ref 130).

20. LOGIC ANALYZER. A computer program used to automatically reconstruct equations forming the basis of a program and to flowchart assembly language programs. One such program translates assembly language instructions into a machine-independent microprogramming language and builds the microprogramming statements into a network in which the flow of control is analyzed and equations are reconstructed (Ref 131).

21. MANAGEMENT INFORMATION SYSTEM. Consists of a computer-based information system (a particular combination of human service, material service, and equipment service) for the purpose of gathering, organizing, communicating, and presenting information to be used by individuals for planning and controlling an enterprise (Ref 132). Several packages which serve as essential elements of a management information system are discussed with examples in a recent publication (Ref 133).

22. REQUIREMENTS TRACER. A computer program used to provide traceability from requirements through design and implementation of the software products. Traceability is characterized to the extent that an audit trail exists for the successive implementation of each requirement. The University of Michigan-developed Problem Statement Language/Analyzer serves as an example (Ref 134). Experience using the Michigan and other similar systems has been reported (Ref 135).

23. SIMULATOR. A computer program that provides the target system with inputs or responses that resemble those that would have been provided by the process for the device being simulated. The simulator's function is to present data to the system at the correct time and in an acceptable format (Ref 90). Several of the many varieties of simulators available are briefly described as follows:

Interpretive computer simulator: A computer program used to simulate the execution characteristics of a target computer (provides bit-for-bit fidelity with results that would be produced by the target machine) using a sequence of instructions of the host computer.

Peripheral simulator: A computer program used to present functional and signal interfaces representative of a peripheral device to the target system.

Statement-level simulator: A computer program used to simulate the execution characteristics of a target computer at the source instruction-level using a sequence of instructions on the host computer.

System simulator: A mechanization of a model of the system (hardware, software, interfaces) used to predict system performance over time.

24. SOFTWARE MONITOR. A computer program that provides detailed statistics about system performance. Because software monitors reside in memory, they have access to all the tables the system maintains. Therefore, they can examine such things as core usage, queue lengths, and individual program operation to help measure performance. Use of software monitors has been described in a recent publication (Ref 136).

25. STANDARDS. Procedures, rules, and conventions used for prescribing disciplined program development. Architecture and partitioning rules, documentation conventions, language conventions, configuration, and data management procedures, etc., are typical examples under this category (Ref 123).

26. STANDARDS ANALYZER. A computer program used to automatically determine whether prescribed programming standards and practices have been followed. The program

can check for violations to standards set for such conventions as program size, commentary, structure, etc. (Ref 90). The National Bureau of Standards-developed FORTRAN Analyzer serves as an example (Ref 137).

27. STATIC ANALYZER. A computer program used to provide information about the features of a source program. This type of tool examines the source code statically (not under execution conditions) and performs syntax analysis, structure checks, module interface checks, event sequence analysis and other similar functions. Several of the many varieties of static analyzers available are briefly described as follows:

Overlay analyzer: A computer program that examines the source program in order to determine mutually disjoint segments that can reside in the same area of memory at run time (Ref 117).

Units consistency analyzer: A computer program which analyzes the source code version of equations to assure that they consistently reference the global data base (Ref 137).

Usage statistics gatherer: A computer program that computes statistics based on the number of times various items appear in a source program (Ref 117).

28. STRUCTURE ANALYZER. A computer program used to examine source code and determine that structuring rules set for either the control or data structure, or both, have been obeyed. Typically, the program parses an equivalent model of the control or data topology before commencing analysis.

29. TEST BED. A test site composed of actual hardware (hardware test site) or simulated equipment (software test site) or some combination. A hardware test site uses the actual computer and interface hardware to check out the hardware/software interfaces and actual input/output. The program execution is confirmed using actual hardware timing characteristics, but the output is limited and test repeatability is a problem. A software test site uses an instruction-level and/or statement-level simulator to model actual hardware. A software test site permits full control of inputs and computer characteristics, allows processing of intermediate outputs without destroying simulated time, and allows full test repeatability and good diagnostics. The Shuttle Avionics Integration Laboratory represents an elaborate hardware test bed (Ref 138) and the Software Design and Verification System (SDVS) represents a sophisticated software test bed (Ref 139).

30. TEST DRIVERS, SCRIPTS. To run tests in a controlled manner, it is often necessary to work within the framework of a "scenario"--a description of a dynamic situation. To

accomplish this, the input data files for the system must be loaded with data values representing the test situation or events to yield recorded data to evaluate against expected results. These tools permit generation of data in external form to be entered into the system at the proper time (Ref 90).

31. TEST-RESULT PROCESSOR. A computer program used to perform test output data reduction, formatting, and printing. Some perform statistical analysis where the original data may be the output of a monitor (Ref 90).

32. TEXT EDITOR. A computer program used to prepare documentation and perform work-file edits (erase, insert, change, and move words or groups of words). The program requires a facility for on-line storage and recall of text units for inspection, editing, or printing (Ref 90).

33. TIMING ANALYZER. A computer program that monitors and prints execution time for all program elements (functions, routines, and subroutines). A more detailed description of the tool appears in the literature (Ref 131).

APPENDIX I
TOOL SURVEY

(Ref 84)

TOOL SURVEY

A survey of software tools on the candidate target environments was conducted during the first phase of the AMT development. 84 The purpose of the survey was to identify software tools that could be incorporated in the AMT. The survey was limited to the candidate environments because it was felt it was beyond the scope of this effort to transport tools from other environments. The criteria for selection of a tool for consideration for incorporation in the AMT were:

- o Applicability to software measurement (Did the tool provide any metric data?)
- o Portability of tool (Can the tool be used on different hardware configurations?)
- o Interoperability of the tool (How many modifications to the tool are necessary?)
- o Usability of the tool (How much effort is required to learn how to operate the tool? How much effort is there to preparing input and interpreting output that was tool-driven?)

The results of this Tool Survey are presented in matrix form. Background information and analysis of the state-of-the-art of software tools and their applicability to metric appear before it, preceding the Tools Survey. As a result of this analysis, selective tools that have compatible hardware/operating systems with the target environments are also included in the matrix. Finally, the last part deals with the actual tools to be used in the AMT were considered for use, or were applied during its development.

CODING AND IMPLEMENTATION: METRICS APPLICABILITY

The origin of code inspection was structured programming and allied software engineering technologies of the early 1970's. The goal of automated static analysis/evaluation has been to automate the compliance with the techniques and make a search of program properties.

The program parameters are structure-based (program logical and data structure, naming conventions, documentation conventions, etc.), control/data flow based (avoidance of undue control complexity; assurance of well-definedness of variables, etc.), and interface based (assurance of correspondence between modules, subsystem, inter-system, etc.). The anomaly-detecting metrics have to do with standards enforcement (deficiencies in source code), whereas the predictive metrics quantify the logic of design and implementation.

For example, the JOVIAL Automated Metric System (JAMS) is designed to collect structural information about JOVIAL programs. GE's Integrated Software Development System (ISDS) provides a capability to analyze other languages including FORTRAN, PDL, IFTRAN, and PASCAL. A major subsystem of ISDS, the generalized parser (GNP), the grammar description language (GDL) and grammar tables, provides this capability and will be used in the AMT.

Symbolic evaluation of code has as its goal the "interpretation" of program behavior at the programming language level. Assumption must be made about the environment, the deterministic properties of the programming language behavior, and the outcome of symbolic execution results. On systems such as DISSECT or MACSYMA the user interactively chooses a path and performs symbolic interpretation of actions along the chosen path. The system then displays the "formulas" to the user. The user compares original and implemented formulas for equality. Differences between computed and actual formulas are mistakes. Special formula formatting methods are used to make these differences highly visible. Final control software is not yet available. Symbolic evaluation has good candidate potential for the accuracy metrics at the system level.

The final type of static analysis tools, proof of correctness, can be used at the system level, subsystem level, or the module level as assessments of different levels of correctness. The Failure of Proof Method (FPM), uses a mathematical approach to proving the correspondence between a program and its formal specification. The consistency metric is highly visible here.

Dynamic testing is achieved through system exercising of programs. Typical self-testing metrics for higher level language systems have been built on a experimental basis and include:

- o Automatic specified percentage of program logical segment coverage in any one test; aggregated test coverage of close to 100%.

- o Assistance in setting input values and evaluating output values.
- o Some form of automated results comparison.

These dynamic test tools consist of two basic modules, an instrumentation module and an analyzer module. The source language program is submitted directly to the instrumentation module. Then the instrumentation module accepts the source program of the module under test and instruments it by inserting additional statements in the form of counters or sensors. The instrumented source file is compiled and executed. At this point an analyzer module produces a report documenting the behavior under the test during its execution.

Typical metric-like data reported are:

- o Max and min values of variables.
- o Number and percentage of subroutine calls executed.
- o Measures of program complexity.
- o Statement consistency checks.
- o Program cross-references.
- o Trace capability.
- o Flagging of non-ANSI code.
- o Logically impossible - path detection.
- o Subroutine argument/parameter verification.
- o Data range check.
- o If statement trace.
- o Branch trace.
- o Subroutine/statement timing.
- o Min/max assignment values.
- o First/last assignment values.
- o Min/max DO Loop Control Variable.
- o Final DO Loop Index Value.
- o Final branch values.

- o Statement, path, segment, module interface or flow execution frequencies.
- o Specific data associated with each executable source statement.
- o Subroutine retrace capability, complete calling tree, reverse execution capability.
- o Performance indices for modules and input data.

A list of dynamic tools would include: JAVS, CABS, FAVS, RXVP, FORTUNE, CIP, FORSAP, FETE, PROGFORT, PROGTIME, TPL, and TAP.

The goal of mutation analysis is to show that small changes in program are discovered by test data. Conversely, the test data must be strong enough to catch the significant errors. Relevance to error detection metrics is obvious.

The Pilot Mutation System (PIMS) has been applied to FORTRAN and COBOL pilot systems. Magnitude of the mutant error is classified as:

- o Program does not compute.
- o Program computes but does not run test data.
- o Program compiles, test run is satisfactory, and the program is either logically equivalent to the original or test data is not good enough.

Reliability analysis is still in its infancy. The goal is to determine whether all defects have been reliably removed by tests. Any error must be made known by some combination of inputs. Following this theoretical approach of examining all possible input combinations is prohibitive in terms of cost effectiveness and computer time/capacity. The Next Error Discovery Prediction method fails because software reliability simply does not follow the probability laws of hardware reliability.

MATRIX OF SOFTWARE TOOLS

The matrix of software tools having potential metric applicability follows. It includes tools currently in use or planned for at RADC and additional non-RADC tools also worthy of consideration for AMT development or usage.

MATRIX OF SOFTWARE TOOLS HAVING MILITARY APPLICATIONS

TOOL OR SYSTEM NAME	FUNCTIONS	SOURCE	IBM OS VERSIONS	LANGUAGE PROCESSED	IMPLEMENTATION LANGUAGE	REFERENCES	STATUS	CONTACT
1. Tools currently in use or planned use at RADC								
1. Software Science Analyzer	Accepts COBOL source as input & produces S/W science parameters	Purdue University	PIP 11/70, Purdue Mac & IAS	COBOL	COBOL	"Elements of Software Science", 1977	Planned	
2. MUC	Automated tool which will enforce & produce established programming standards	Softech	IBM 360/370	JOVIAL J73	JOVIAL 73	MITRE Tech Report	Operational	Mr. Richard Mottu RADC/ISIS
3. Basic Statistics Collector	Performs static analysis of BASIC programs (ie, counts statements, loop nesting, characters/line, etc.)	RADC/ISIS	IBM, Multics	PL/I	PL/I, BASIC	Basic Statistics Collector RADC-TR-76-9	Operational for PL/I BASIC (planned)	Mr. Douglas White RADC/ISIS
4. Comm Software Development Package	Assists in the production of programs for communications applications	LSC	IBM, Multics	JOVIAL J3	JOVIAL J73 PL/I		Operational	
5. Data & Analysis Center for Software (DACCS)	Source of empirical data on S/W dev and maintenance for RADC	RADC/ISIS	IBM 80 GCDS				Operational	Mr. John Palumbo RADC/ISIS
6. Debug & Test Microprogram Tools	a) Global Cross Reference Analyzer b) Control Flow Analyzer c) Data Flow Analyzer d) Timing Analyzer e) Source Code Comparator	US. Air Force		N/A	N/A	Reliability Programming, RADC-TR-79-173 CDD Manual 4120.17M	Operational	Mr. Donald Roberts, RADC/ISIS

Figure 17

MATRIX OF SOFTWARE TOOLS HAVING METRIC APPLICATIONABILITY

TOOL OR SYSTEM NAME	FUNCTIONS	SOURCE	IBM OS VERSIONS	LANGUAGE PROCESSED	IMPLEMENTATION LANGUAGE	REFERENCES	STATUS	UNIT
b. (CUMI)	f) Symbolic Executor g) Test Case Generator h) Microprogram Execution Monitor	U.S. Air Force		N/A	N/A	Reliability Programming, RADC-TR-79-173, UDO Manual 4120, 17M	Operational	Mr. Donald Roberts, RADC/1515
7. JUVIAL/J3 Statistics Collector	Counts, averages, & % of source program by type of data used; size & nature of arrays, types of statements, comments; DEFINE directives	U.S. Air Force	HOMEWELL 600/6000 GCOS	JOVIAL/J3	JOVIAL/J3	JUVIAL/J3 Statistics Collector, RADC-TR-77-293	Operational	Mr. Richard Slavinski, RADC/1515
d. JUVIAL Automated Verification System (JAVS)	Recognition of untested program paths, develop additional test cases, document the computer program with metric - like counters and sensors	U.S. Air Force	H6180 GCOS	JOVIAL	JOVIAL J3	JAVS Final Report, RADC-TR-78-247; JAVS Technical Report, RADC-TR-77-126	Operational	Mr. Frank La Monica, RADC/1515
9. JOVIAL Compiler Validation Systems (JCVS)	Test Compiler	U.S. Air Force		JOVIAL J73	JOVIAL J73	JCVS, RADC-TR-74-232; JCVS User's Guide, RADC-TR-73-268	Being Developed	Mr. R.I. Slavinski, RADC/1515
10. FORTRAN Automated Verification System (FAVS)	Static detection of unreachable statements, set/use errors, mode conversion errors, & external references errors; test case development & source code instrumentation; automated documentation	U.S. Air Force	H6180 GCOS	FORTRAM	DFMTRAM	FAVS, RADC-TR-78-268	Operational	Mr. Frank La Monica, RADC/1515

Figure 17 (continued)

MATRIX OF SOFTWARE TOOLS HAVING METRIC APPLICABILITY

UNIQUE SYSTEM NAME	FUNCTIONS	SOURCE	HW DS VERSIONS	LANGUAGE PROCESSED	IMPLEMENTATION LANGUAGE	MILESTONES	STATUS	CURATOR
11. FORTMAN Code Auditor	Automated documentation, format, design, & structural standards	U.S. Air Force	H6180	FORTMAN	FORTMAN	FORTMAN Code Auditor, RADC-TR-76-395	Operational	Mr. Frank La Monica RADC/ISIS
12. Semantics Oriented Language (SEMANOL)	Applied to proposed or implemented higher order language & will detect any inconsistencies in the HOL specifications	U.S. Air Force	HIS 600/6000 Multics	SEMANOL	JOVIAL/J3, JOVIAL/J73, CMS - 2, BASIC	SEMANOL RADC-TR-75-711, Improvements to SEMANOL RADC-TR-77-305	Operational	Mr. John White RADC/ISIS
13. AVIONICS Software Reliability Prediction Model	Prediction of the reliability and mean time to failure of the software development project	U.S. Air Force	H6180 Multics			Final document not yet published	Operational	Mr. Alan Sukert RADC/ISIS
14. MACCS Program Support Library (PSL)	PSL provides support & records of all aspects of the program development process including design, coding, testing, documentation, & maintenance	U.S. Air Force	HONEYWELL H6000/MACCS	COBOL	ANG1 COBOL and GMP		Operational	Mr. Lawrence Lombardo RADC/ISIS
15. COBOL Structured Programming Precompiler	Additions to COBOL, X.3.23-1968, are in the form of structuring verbs which permit the programmer to write the basic control logic figures required to implement structured programming forms.	U.S. Air Force	FPM 370 & HONEYWELL H6180	COBOL	COBOL	Structured Programming Series RADC-TR-74-300	Operational	Mr. Frank La Monica RADC/ISIS

Figure 17 (continued)

IBM IM SYSTEM NAME	FUNCTIONS	SOURCE	I/W OS VERSIONS	LANGUAGE PROCESSED	IMPLEMENTATION LANGUAGE	REFERENCES	STATUS	CONTACT
15. LUBWA Automated Verification System (CAVS)	Integrated, automated Test	U.S. Air Force	IBM 370	COROL	COROL		Planned	Mr. Frank La Monte NAAC/ISIS
Additional Software Tools to Consider Currently Not Listed as Being Used at the Target AMT Facilities								
1. Problem Statement Language/Problem Statement Analyzer (PSL/PSA)	requirements documentation & analysis	Univ. of Michigan	IBM 370 H 6000 CDC 6000 V 1100	Program Specification Language	FORTRAN	PSL/PSA User's Guide	Operational, available	D. Leichner Univ. of Michigan
JAMS	JOVIAL code analysis	GE	PDP 11/40 RSX11M	JOVIAL 34	FORTRAN	IRAD Final Report	prototype	Jim McCall GE
3. DISSECT & HASYMA	symbolic code analysis	Univ. of Massachusetts Univ. of CA, San Diego				IEEE Trans. Software/77 Engineering	Research aid	L. Clark, Univ. of Mass., W. Haden, Univ. of CA
4. Proof of Correctness Method (PCM)	mathematical approach to proving the correspondence between a program and its formal specification					IEEE Trans. Software Engineering 9/76	Research aid	

Figure 17 (continued)

THEIR SYSTEM NAME	FUNCTIONS	SUMMIT	H/W OS VERSIONS	LANGUAGE PROCESSED	IMPLEMENTATION LANGUAGE	REFERENCES	STATUS	COMMENTS
a. DYNAMIC TESTING OF CODE	Integrated, automated testbed to gather statistics of the program code.	GRC	MS-DOS	FORTRAN	IFTRAM	GRC Manuals	operational	GRC
b. FORTUNE		CAPEX	IBM 360/OS	FORTRAN	FORTRAN	User Manual WARDAC, Washington, D.C. #8850128, IN112	operational	Naval Data Automation Command
c. CIP		WARDAC SYSTEMS SUPPORT	IBM 360/OS	COMAL	COMAL		operational	
d. FORTAPAPAPS		UCLA	IBM 360/OS	FORTRAN	FORTRAN		research aid	
e. FELI		STANFORD	IBM 360/OS	FORTRAN	FORTRAN		research aid	
f. PROKFOR		STANFORD	IBM 360/OS	FORTRAN	FORTRAN		research aid	
g. PROKTIME		STANFORD	IBM 360/OS	FORTRAN	FORTRAN		research aid	
h. TPL		U-CRSD	POP-11/45 RSX-11C MS-DOS, GCOS 11E CPC 6400	FORTRAN	FORTRAN	Test Procedures D.T. Panzi	operational	U. Panzi CRU, U
i. IAP		GRC	IB5000	FORTRAN	IFTRAM		operational	
b. Pilot Mutation System (PIMS)	Makes small modifications (mutations) in original program to produce a mutant, then classify type of result. Error seeding	Univ. of Calif., Berkeley, Georgia Institute of Technology				Georgia Institute of Technology, Report GIT-ICS-79/08	prototype	K.J. Lipton University of Calif. Berkeley, Ca

Figure 17 (continued)

TOOL OR SYSTEM NAME	FUNCTIONS	SOURCE	H/W OS VERSIONS	LANGUAGE PROCESSED	IMPLEMENTATION LANGUAGE	REFERENCES	STATUS	CONTACT
(LUMI) 7. Integrated Software Development System (ISDS)	Provides structural information about program code	GE	PDP11, 28K DOS/BATCH, RSX-110, VERSATEC P/P, TEKTRONIX 4012	FORTRAM, PDL, IFTRAM, PASCAL	IFTRAM/Available in FORTRAM	TIS76CIS01	operational	Gene Walter, GE, Sunnyvale

Figure 17 (continued)

TOOL	DESCRIPTION OF TOOL
<u>TO BE USED IN AMT:</u> - ISDS GNP - ISDS UTLCMP - H6000 GCOS UTLGTC	Generalized Parser - for structural analysis Compare strings of characters Obtain user enter command
<u>CONSIDERED FOR USE IN AMT:</u> - Software Science Analyzer (Purdue)	From COBOL source produces software science parameters.
- COBOL Usage Analyzer (Texas A&M)	Software tool for measuring usage of COBOL language by program.
- CAVS (GRC)	COBOL Automated Verification System
- COBOL Structured Pre-compiler	Structuring verbs allow for the implementation of structured programming forms.
- PSL/PSA	Problem Statement Language/Problem Statement Analyzer
<u>USED ON AMT:</u> - SPDL (GE)	Structured Program Design Language
- IFTRAN (GRC)	Structured FORTRAN
- ISDS (GE)	Integrated Software Development System

Figure 18. TOOL USAGE

APPENDIX J
PROCEDURES FOR ASSESSING SOFTWARE QUALITY
(Ref 28)

APPENDIX J

PROCEDURES FOR ASSESSING SOFTWARE QUALITY

The benefits of applying the software quality metrics are realized when the information gained from their application is analyzed. The analyses that can be done based on the metric data are described in the subsequent paragraphs. There are three levels at which analyses can be performed. These levels are related to the level of detail to which the quality assurance organization wishes to go in order to arrive at a quality assessment (Ref 28).

Inspector's Assessment

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

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

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

Sensitivity Analysis

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

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

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

$$M_d^m = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1k} \\ m_{21} & & & \\ \vdots & & & \\ m_{n1} & & & m_{nk} \end{bmatrix}$$

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

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

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

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

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

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

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

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

Where T_i = threshold value
specified for metric i

Use of Normalization Function to Assess Quality

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

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

$$C_d^m = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ . & . & & . \\ . & . & . & . \\ . & . & . & . \\ c_{l1,1} & & & c_{l1,n} \end{bmatrix}$$

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

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

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

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

$$CM = R_d^m = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1,k} \\ . & . & & . \\ . & . & . & . \\ . & . & . & . \\ r_{l1,1} & & & r_{l1,k} \end{bmatrix}$$

This approach represents the most formal approach to evaluating the quality of a product utilizing the software quality metrics.

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

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

To illustrate the procedure the normalization function that has been developed for the factor Flexibility will be used. The normalization function, applicable during the design phase, relates measures of modular implementation to the flexibility of the software. The predicted rating of flexibility is in terms of the average time to implement a change in specifications. The normalization function is shown in Figure 19. The measurements associated with the modular implementation metric are taken from design documents. The measurements involve identifying if input, output and processing functions are mixed in the same module, if application and machine-dependent functions are mixed in the same module and if processing is data volume limited. As an example, assume the measurements were applied during the design phase and a value of 0.65 was measured. Inserting this value in the normalization function results in a predicted rating for flexibility of .33 as identified by point A in Figure 19. If the Development Manager had specified a rating of 0.2 which is identified by point B, he has an indication that the software development is progressing well with respect to this desired quality.

Calculate Confidence in Quality Assessment. Using statistical techniques a level of confidence can be calculated. The calculation is based on the standard error of estimate for the normalization function and can be derived from a normal curve table found in most statistics texts. An example of the deviation process is shown in Figure 20 for the situation described above. Here it is shown that the Development Manager has an 86 percent level of confidence that the flexibility of the system will be better than the specified rating.

Reporting Assessment Results

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

Table XXI. Normalization Functions

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

(Continued)

Table XXI. Normalization Functions (Continued)

FLEXIBILITY (IMPLEMENTATION)		
MULTIVARIATE FUNCTION	.22 ^M MO.2 ⁺ .44 ^M GE.2 ⁺ .09 ^M SD.3	
INDIVIDUAL FUNCTIONS	.6 ^M MO.2 .72 ^M GE.2 .59 ^M SD.2 .56 ^M SD.3	MO.2 Modular Implementation Measure GE.2 Generality Checklist SD.2 Effectiveness of Comments Measure SD.3 Descriptiveness of Implementation Language Measure
PORTABILITY (IMPLEMENTATION)		
MULTIVARIATE FUNCTION	-1.7 ⁺ .19 ^M SD.1 ⁺ .76 ^M SD.2 ⁺ 2.5 ^M SD.3 ⁺ .54 ^M MI.1	
INDIVIDUAL FUNCTIONS	1.07 ^M SI.1 1.1 ^M MI.1 1.5 ^M SD.2	SD.1 Quantity of Comments SD.2 Effectiveness of Comments Measure SD.3 Descriptiveness of Implementation Language Measure MI.1 Machine Independence Measure SI.1 Design Structure Measure

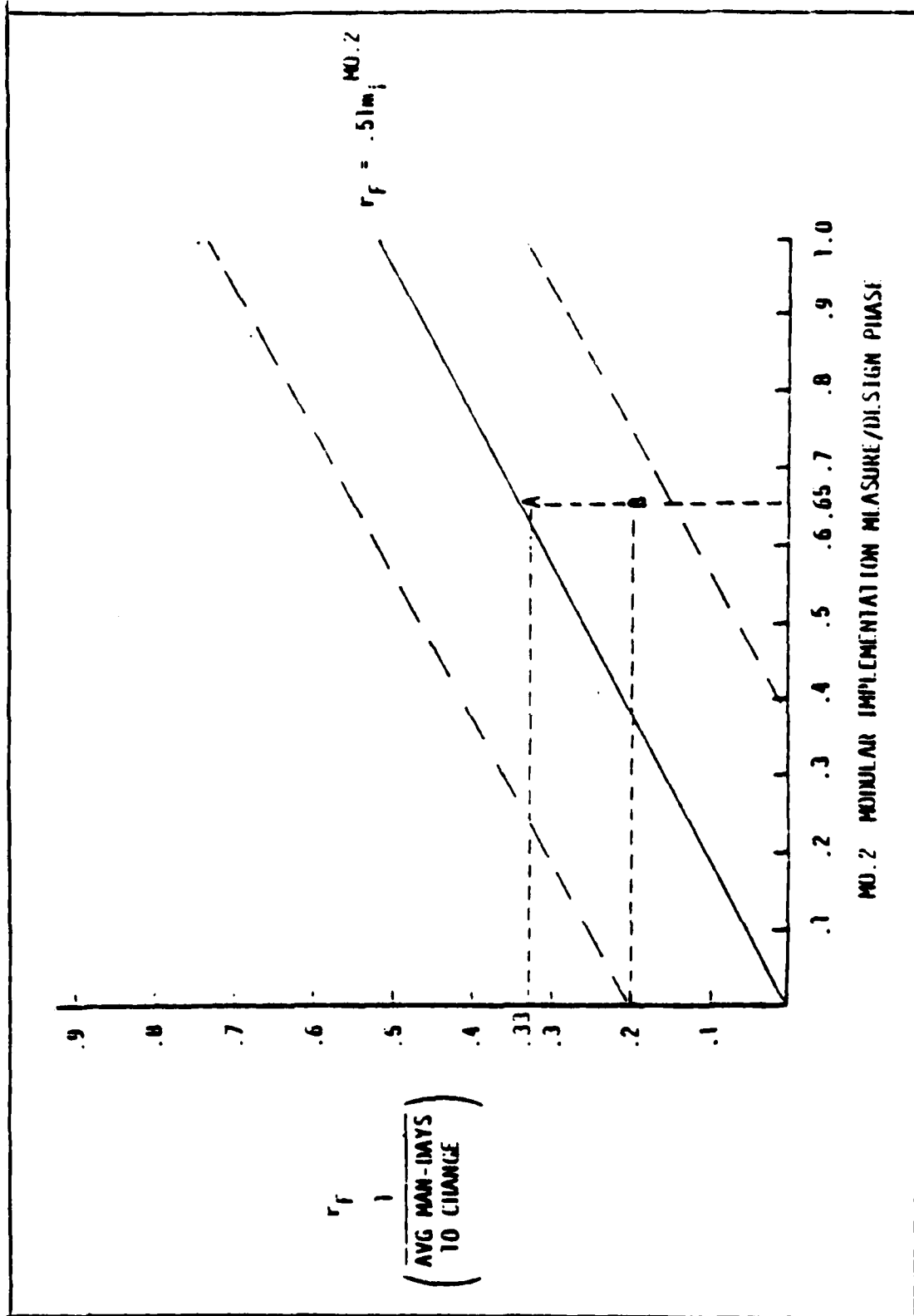


Figure 19. Normalization Function for Flexibility During Design

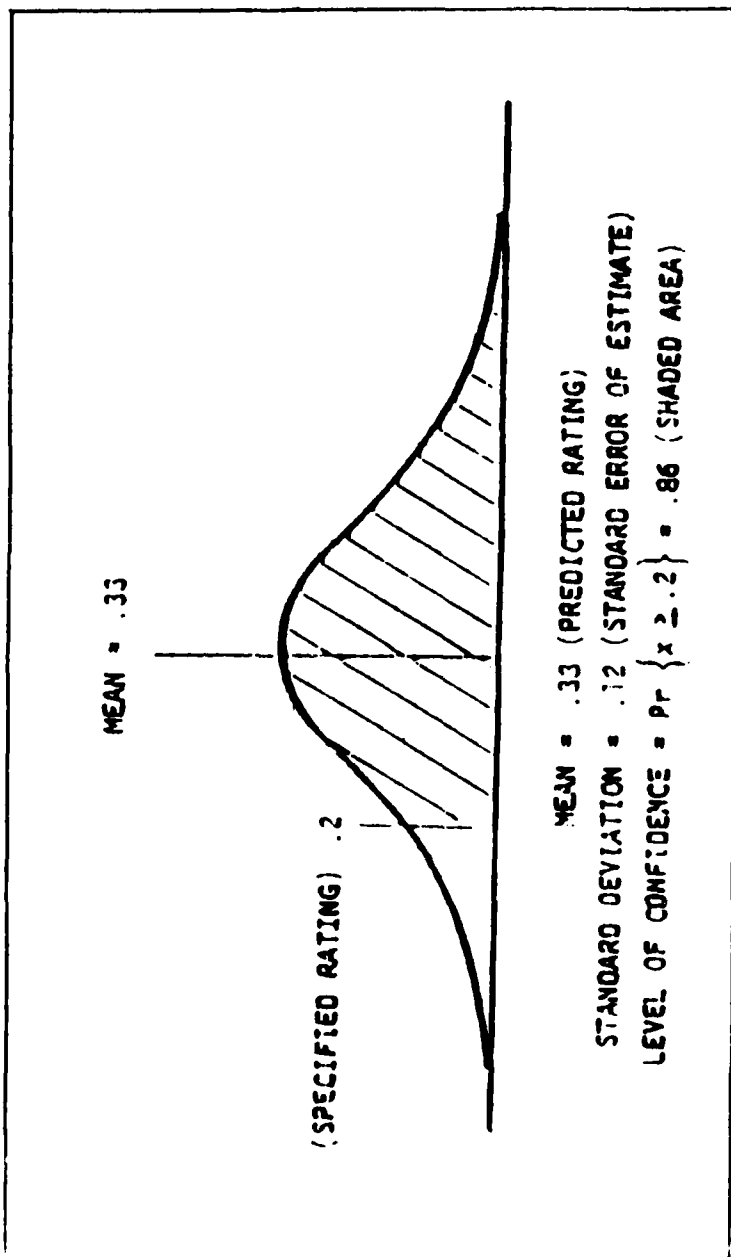


Figure 20. Determination of Level of Confidence

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

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

<u>QUALITY GOALS</u>	<u>PREDICTED RATING BASED ON DESIGN DOCUMENT</u>
RELIABILITY .9	.8
MAINTAINABILITY .8	.95

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

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

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

APPENDIX K
SAMPLE OF AMT REPORTS
(Ref 88)

WORKSHEET REPORT

The worksheet report displays the raw data entered in each worksheet. It represents the current values in the data base. It is used to verify and track data entry.

AUTOMATED MEASUREMENT TOOL

WORKSHEET REPORT

WORKSHEET 3

DATA Base Antexs
MODULE: EXSGET

DATE: 12/23/81

I. STRUCTURE (RELIABILITY, MAINTAINABILITY, TESTABILITY)

1. NUMBER OF LINES OF CODE	95.
2. NUMBER OF LINES EXCLUDING COMMENTS	47.
3. NUMBER OF MACHINE LEVEL LANGUAGE STATEMENTS	0.
4. NUMBER OF DECLARATIVE STATEMENTS	4.
5. NUMBER OF DATA MANIPULATION STATEMENTS	5.
6. NUMBER OF STATEMENT LABELS (EXCLUDING FORMAT STATEMENTS)	0.
7. NUMBER OF ENTRANCES INTO MODULE	1.
- ENTER [CR] TO CONTINUE 'E' TO EXIT:

8. NUMBER OF EXISTS FROM MODULE	2.
9. MAXIMUM NESTING LEVEL	3.
10. NUMBER OF DECISION POINTS (IF, WHILE, REPEAT, DO, CASE)	10.
11. NUMBER OF SUB-DECISION POINTS	0.
12. NUMBER OF CONDITIONAL BRANCHES (COMPUTED TO GO	6.
13. NUMBER OF UNCONDITIONAL BRANCHES (GOTO, ESCAPE)	0.
14. NUMBER OF LOOPS (WHILE, DO)	4.
15. NUMBER OF LOOPS WITH JUMPS OUT OF LOOPS	0.
16. NUMBER OF LOOPS INDICIES THAT ARE MODIFIED	0.
17. NUMBER OF MODULE MODIFICATIONS (SWITH, ALTER)	0.
18. NUMBER OF NEGATIVE OR COMPLICATED COMPOUND BOOLEAN EXPRESSIONS	0.
19. IS A STRUCTURED LANGUAGE USED?	YES
20. IS FLOW TOP TO BOTTOM (ABSENSE OF BACKWARD BRANCHING GOTO's)?	YES

II. CONCISENESS (MAINTAINABILITY)

1. NUMBER OF OPERATORS	4.
2. NUMBER OF UNIQUE OPERATORS	1.
3. NUMBER OF OPERANDS	8.
4. NUMBER OF UNIQUE OPERANDS	3.
- ENTER (CR) TO CONTINUE, 'E' TO EXIT:

EXCEPTION REPORT

The exception report delivers the relationship of each module to a given threshold value of a particular metric. The relationship (less than, equal to, or greater than) and the threshold value is input from the user. This report can be used to identify modules whose scores do not meet a certain threshold, identifying them as potential problems.

AUTOMATED MEASUREMENT TOOL EXCEPTIONS REPORT

DATABASE: AMTEXS

DATE: 12/23/81

METRIC: ET. 2

PHASE: MODULE IMPLEMENTATION

THRESHOLD VALUE: 0.65

RELATION: LESS THAN

THE FOLLOWING MODULES ARE WITHIN RANGE REQUESTED

<u>MODULE NAME</u>	<u>VALUE</u>
EXSCEX	0.
EXCDLP	0.500
EXSDBG	0.333
EXSHLP	0.
EXSPGR	0.
EXSUPK	0.

NORMALIZATION REPORT

The Normalization Report provides the user with the overall rating of a selected quality factor. A series of regression equations are displayed which have been empirically derived from research. The current metric values are substituted in the equations and a rating for the selected quality factor is calculated. Regression equations exist for the quality factors reliability, maintainability, portability, and flexibility only:

AUTOMATED MEASUREMENT TOOL NORMALIZATION FUNCTION REPORT

DATABASE: AMTEXS

MODULE: EXSGET

DATE: 12/23/81

DESIGN NORMALIZATION FUNCTION

IMPLEMENTATION NORMALIZATION FUNCTION

FACTOR: PORTABILITY

NO DESIGN NORMALIZATION FUNCTION
FOR PORTABILITY FACTOR

PORTABILITY = $-1.7 + .19 (SD.1) +$
 $.76(SD.2) + 2.5(SD.3) + .64(MI.1)$
SD.1 = 0.426
SD.2 = 0.857
SD.3 = 1.000
MI.1 = 0.972

PORTABILITY = 2.154

METRIC REPORT

This report calculates the value of each metric catagorized by factor and by development phase. This report is used to determine a total picture of the project as measurements are taken.

AUTOMATED MEASUREMENT TOOL METRIC REPORT/MODULE IMPLEMENTATION PHASE

DATABASE: AMTEXS

MODULE: EXSGET

DATE: 12/23/81

FACTOR	CRITERIA	METRIC	VALUE
CORRECTNESS	Traceability	TR.1	1.000
	Completeness	CP.1	0.667
	Consistency/Procedure	CS.1	1.000
	Consistency/Data	CS.2	0.500
RELIABILITY	Consistency/Procedure	CS.1	1.000
	Consistency/Data	CS.2	0.500
	Accuracy	AY.1	1.000
	Error Tolerance/Control	ET.1	1.000
	Error Tolerance/Input Data	ET.2	1.000
	Error Tol./Computational Fail.	ET.3	0.
	Design Structure	SI.1	0.625
	Complexity	SI.3	0.100
MAINTAINABILITY	Code Simplicity	SI.4	0.722
	Consistency/procedure	CS.1	1.000
	Consistency/Data	CS.2	0.500
	Design Structure	SI.1	0.625
	Complexity	SI.3	0.100
	Code Simplicity	SI.4	0.722
	Modular Implementation	MO.2	0.750
	Quantity of Comments	SD.1	0.426
	Effectiveness of Comments	SD.2	0.857
	Conciseness	CO.1	1.000
TESTABILITY	Design Structure	SI.1	0.625
	Complexity	SI.3	0.100
	Code Simplicity	SI.4	0.722
	Modular Implementation	MO.2	0.750
	Quantity of Comments	SD.1	0.426
	Effectiveness of Comments	SD.2	0.857
PORTABILITY	Descriptiveness of Impl. Lang.	SD.3	1.000
	Modular Implementation	MO.2	0.750
	Quantity of Comments	SD.1	0.426

FACTOR	CRITERIA	METRIC	VALUE
	Effectiveness of Comments	SD.2	0.857
	Descriptiveness of Impl. Lang.	SD.3	1.000
	System Software/Independence	SS.1	0.500
	Machine Independence	MI.1	0.972
REUSABILITY	Modular Implementation	MO.2	0.750
	Generality/Implementation	GE.2	0.750
	Quantity of Comments	SD.1	0.426
	Effectiveness of Comments	SD.2	0.857
	Descriptiveness of Impl. Lang.	SD.3	1.000
	System Software/Independence	SS.1	0.500
	Machine Independence	MI.1	0.972
FLEXIBILITY	Modular Implementation	MO.2	0.750
	Generality/Implementation	GE.2	0.750
	Data Storage Expansion	EX.1	0.
	Computational Extensibility	EX.2	0.500
	Quantity of Comments	SD.1	0.426
	Effectiveness of Comments	SD.2	0.857
	Descriptiveness of Impl Lang.	SD.3	1.000
INTEROPERABILITY	Modular Implementation	MO.2	0.750
EFFICIENCY	Iterative Processing	EE.2	1.000
	Data Usage	EE.3	0.668

STATISTICS REPORT

The Statistics Report provides a profile of COBOL constructs for each module.

AUTOMATED MEASUREMENT TOOL STATISTICS REPORT

DATABASE: AMTEXS

MODULE: EXSGET

DATE: 12/23/81

NUMBER OF LINES OF CODE	95.
NUMBER OF PERFORM STATEMENTS	4.
NUMBER OF EXTERNAL CALLS	0.
NUMBER OF EXECUTABLE STATEMENTS (PROCEDURE DIVISION)	43.
NUMBER OF COMMENTS	48.
NUMBER OF DECLARATIONS (DATA DIVISION)	4.
NUMBER OF LABELS	0.
NUMBER OF I/O REFERENCES	6.
NUMBER OF REDEFINES (EQUIVALENTS)	0.
NUMBER OF LEVEL 88 DATA ITEMS (LOCAL VARIABLES)	1.

SUMMARY REPORT

The summary report provides a summary of the metric scores for all of the modules in the system.

AUTOMATED MEASUREMENT TOOL METRIC SUMMARY REPORT

DATABASE: AMTEXS

DATE: 12/23/81

MODULE: EXSGET

AY.1 = 1.000	CO.1 = 1.000	CP.1 = 0.667	CS.1 = 1.000
CS.2 = 0.500	EE.2 = 1.000	EE.3 = 0.668	ET.1 = 1.000
ET.2 = 1.000	ET.3 = 0.	EX.1 = 0.	EX.2 = 0.500
GE.2 = 0.750	MI.1 = 0.972	MO.2 = 0.750	SD.1 = 0.426
SD.2 = 0.857	SD.3 = 1.000	SI.1 = 0.625	SI.3 = 0.100
SI.4 = 0.722	SS.1 = 0.500	TR.1 = 1.000	

QUALITY GROWTH REPORT

When the user wishes to track the value of a particular metric over time, the Quality Growth Report will furnish a tabular display of the scores of a selected metric over the phases of the project. This report is used to track a particular metric through a project to see how its value changes.

AUTOMATED MEASUREMENT TOOL QUALITY GROWTH REPORT

DATABASE: AMTEXS
MODULE: EXSGET

DATE: 12/23/81

METRIC	DETAILED DESIGN	MODULE IMPLEMENTATION
ET.2	0.750	1.000

MATRIX REPORT

This report displays the average and standard deviations for all metric values modules. This report displays all of this information in a matrix form allowing the user to easily identify modules with metric scores that vary from the system average.

AUTOMATED MEASUREMENT TOOL MATRIX REPORT

DATABASE: AMTEXS

PAGE = 1

DATE: 12/23/81

MODULE NAME	AY.1	CO.1	CP.1	CS.1	CS.2	EE.2
EXSCEX	0.	1.000	0.	0.	0.	1.000
EXSCHK	1.000	1.000	0.667	1.000	0.500	1.000
EXSCLP	1.000	1.000	0.667	1.000	0.500	1.000
EXSDBG	0.	1.000	0.	0.	0.	0.
EXSGET	1.000	1.000	0.667	1.000	0.500	1.000
EXSHLP	0.	1.000	0.833	1.000	0.500	0.
EXSPGR	0.	1.000	1.000	1.000	0.500	1.000
EXSQRY	0.	1.000	0.667	1.000	0.500	0.
EXSSSM	0.	1.000	1.000	1.000	0.500	0.
EXSUPK	0.	1.000	0.625	1.000	0.500	1.000
AVERAGE =	0.300	0.900	0.550	0.700	0.350	0.500
STANDARD DEVIATION =	0.438	0.316	0.401	0.483	0.242	0.527

MODULE REPORT

This report displays the catalog of modules that have been entered into the database. It provides a status report on the database.

AUTOMATED MEASUREMENT TOOL MODULES REPORT

DATABASE: AMTEXS

DATE: 12/23/81

WS1 CONTAINS SOME NIL VALUES

WS2A CONTAINS SOME NIL VALUES

THE FOLLOWING MODULES ARE PRESENTLY IN THE CURRENT DATABASE:

1. EXSCEX **	2. EXSCHK *
3. EXSCLP *	4. EXSDBG **
5. EXSGET *	6. EXSHLP *
7. EXSPGR *	8. EXSQRY *
9. EXSSSM *	10. EXSUPK *

TOTAL NUMBER OF MODULES IN DATABASE IS 10.

NOTE: * INDICATES BOTH WS2B AND WS3 CONTAIN SOME NIL VALUES.

NOTE: ** INDICATES WS2B CONTAINS SOME NIL VALUES.

VITA

Stanley J. Jarzombek, Jr. was born on 27 May 1955 in McAllen, Texas. He graduated from Sharyland High School in Mission, Texas in 1973 and attended the University of Texas at Austin for one year prior to enlisting in the USAF in August 1974. He served three years in personnel. Promoted below-the-zone, he made Sergeant at one year and seven months. Selected for ASCP, he returned to the University of Texas to complete his undergraduate education. He received a B.A. in Computer Science and a B.B.A. in Data Processing and Analysis. Upon graduation, he received his commission in the USAF as a Distinguished Graduate through the ROTC program. He then entered the School of Engineering, Air Force Institute of Technology, in August 1980.

Permanent address: Route 2, Box 1592-f

McAllen, Texas 78501

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/MA/82M-1	2. GOVT ACCESSION NO. AD-A115 501	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SOFTWARE QUALITY METRICS: A SOFTWARE MANAGEMENT MONITORING METHOD FOR AFLC IN ITS SOFTWARE QUALITY ASSURANCE PROGRAM FOR THE QUANTITATIVE ASSESSMENT OF THE SYSTEM DEVELOPMENT LIFE CYCLE UNDER CONFIGURATION MANAGEMENT		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis
7. AUTHOR(s) Stanley J. Jarzombek, Jr., 2nd Lt, USAF		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS School of Engineering Air Force Institute of Technology, WPAFB, OH 45433		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Department of Mathematics AFIT/ENC, WPAFB, OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE March 1982
		13. NUMBER OF PAGES 327
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) 15 APR 1982 Dean for Research and Professional Development Air Force Institute of Technology (ATC) Wright-Patterson AFB, OH 45433		
18. SUPPLEMENTARY NOTES APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17 <i>Fred C. Lynch</i> FREDRIC C. LYNCH, Major, USAF Director of Public Affairs		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
SOFTWARE QUALITY ASSURANCE	SOFTWARE METRICS	DECISION-MAKING
SYSTEM DEVELOPMENT LIFE CYCLE	SOFTWARE TOOLS	SOFTWARE QUALITY METRICS
CONFIGURATION MANAGEMENT	SQA TECHNIQUES	SOFTWARE ATTRIBUTES
PROJECT MANAGEMENT	METRICS	SOFTWARE FACTORS
AUTOMATED MEASUREMENT	QUALITY ASSURANCE	MAINTAINABILITY
		RELIABILITY
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Software Quality Assurance (SQA) is recognized as an essential function needed to monitor the software system development life cycle (SDLC). The framework established for Software Quality Metrics (SQM) provides goal-directed system specifications and the ability to quantitatively assess the quality of the system under development. The Automated Measurement Tool (AMT), which operationalizes the application of SQM, functions as the core of a Decision Support System, providing quantitative measures and various levels of reports. A literature survey of SQA aids enabled the recommendation of a minimum		

Cont
set of tools and techniques to be used by the SQA program for monitoring the SDLC, which has been envisioned as an iterative process controlled by management. Recognizing the functional impact of specific information as the key to objectively monitoring and controlling the software system development, the decision-making model was conceptualized as three subsystems within each phase of the SDLC: scanning (afferent), organizing (intelligence), and decision (efferent). The use of checklists by system developers highlights a prescriptive method of goal-directed development. The thesis provides justification for using SQM by reviewing the need and demonstrating how the concepts can now be used.

R