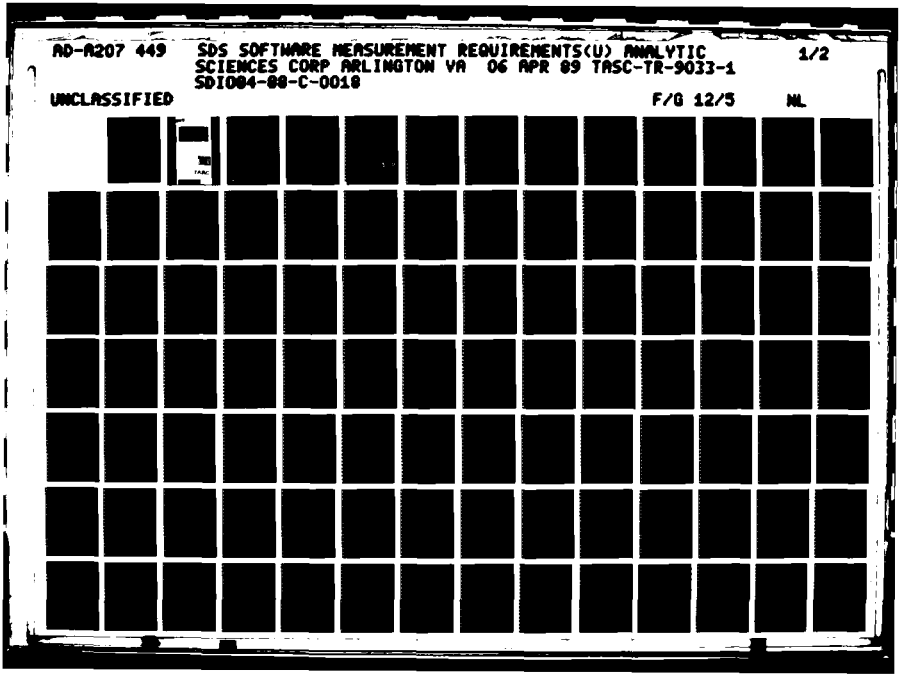
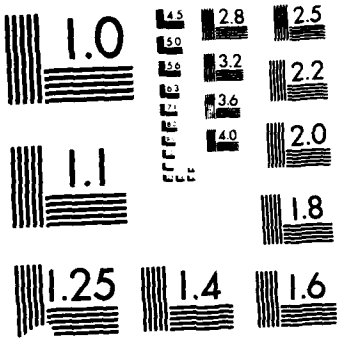
AD-A207 449 SDS SOFTWARE MEASUREMENT REQUIREMENTS(U) ANALYTIC SCIENCES CORP ARLINGTON VA 06 APR 89 TASC-TR-9033-1 SDIO84-88-C-0018 1/2

UNCLASSIFIED F/G 12/5 NL





UTION TEST CHART

(2)

RADING SCHEDULE	Distribution Unlimited
ON REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT
RGANIZATION es Contractor)	6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATIO Strategic Defense Initiative
ZIP Code) t 9	7b. ADDRESS (*City, State, and ZIP Code)* Room 1E149 The Pentagon Washington, D.C. 20301-7I
NSORING nitiative	8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFIC

TASC
THE ANALYTIC SCIENCES CORPORATION

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for Public Release
Distribution Unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
TR-9033-1

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
The Analytic Sciences Corporation (Prime Contractor)

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION
Strategic Defense Initiative Organization

6c. ADDRESS *(City, State, and ZIP Code)*
1700 N. Moore Street
Suite 1800
Arlington, VA 22209

7b. ADDRESS *(City, State, and ZIP Code)*
Room 1E149
The Pentagon
Washington, D.C. 20301-7100

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
Strategic Defense Initiative Organization

8b. OFFICE SYMBOL *(If applicable)*

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State, and ZIP Code)*
Room 1E149
The Pentagon
Washington, D.C. 20301-7100

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
SDS Software Measurement Requirements (U)

12. PERSONAL AUTHOR(S)

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 5Dec88 TO 6Apr89	1989, April, 6	90

16. SUPPLEMENTARY NOTATION Task Report No. TR-9033-1
Prepared by SPARTA, Inc, Teledyne Brown Engineering, and The Analytic Sciences Corporation

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
SDS SOFTWARE, EVALUATION, MEASUREMENT, PROCESS, METRICS, MEASUREMENT PLAN (space defense system; space based; defense systems; (KT))

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This report addresses the criteria for evaluation of SDS software and measurement requirements by identifying the standards and attributes required for SDS software products and the software development process. The identification of measurement requirements for SDS software was performed into three phases. The first phase was a review of standards and development process relevant to SDS, as they pertained to software metrics. The second phase was a detailed examination of SDS software characteristics, definition of software domains, and quality and productivity measurement requirements. The third phase integrated the measurement requirements identified previously into a methodology for the application of software metrics. (Key words)

The software measurement is defined as the act of capturing metrics and comparing them to standards. A metric is defined as a quantitative standard of measurement used to represent and compare some software process or product attribute. The primary objective of software metrics is to predict, throughout the development phase of the software, the quality and overall schedule and cost of the final product. (see reverse)

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☒ DTIC USERS

21 ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
Charles F. Tilley

22b. TELEPHONE *(Include Area Code)*
(202) 693-1600

22c. OFFICE SYMBOL
SDIO/S/PF

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. ABSTRACT (continued)

The process was begun with the identification of potentially relevant standards and development guidelines in order to define the software quality and productivity factors relevant to SDS software. This identification spanned the software development life cycle from requirement definition through operation and maintenance. The methodology requirements were then defined for incorporation of software measurement into the software development process.

(2)

TR-9033-1

TASK ORDER 33
SOFTWARE MEASUREMENT PROCESS

SDS Software Measurement Requirements Technical Report

6 April 1989

Prepared Under:

Contract No. SDIO84-88-C-0018

Prepared For:

STRATEGIC DEFENSE INITIATIVE ORGANIZATION
Office of the Secretary of Defense
Washington, D.C. 20301-7100

Prepared By:

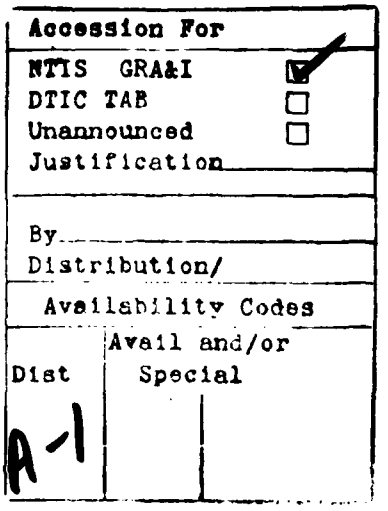
SPARTA, Inc.
Teledyne Brown Engineering
The Analytic Sciences Corporation

THE ANALYTIC SCIENCES CORPORATION
1700 North Moore Street
Suite 1800
Arlington, VA 22209

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)



LIST OF FIGURES

EXECUTIVE SUMMARY

Purpose and Scope

The work presented in this report was performed under Subtask 1 of Task Order 33 of the SDIO Systems SETA contract. The purpose of the subtask was to develop SDS software measurement requirements by identifying the standards and attributes required for SDS software products and the software development process. SPARTA was the technical lead on the subtask, with support from Teledyne Brown and TASC.

The subtask was broken into three phases. The first phase was a review of standards and development process relevant to SDS, as they pertained to software metrics. The second phase was a detailed examination of SDS software characteristics, definition of software domains, and quality and productivity measurement requirements. The third phase integrated the measurement requirements identified previously into a methodology for the application of software metrics.

Standards and Process Identification

For purposes of this document, software measurement is defined as the act of capturing metrics and comparing them to standards. A metric is defined as a quantitative standard of measurement used to represent and compare some software process or product attribute. The primary objective of software metrics is to predict, throughout the development phase of the software, what the quality and overall schedule and cost of the final product will be. The process was begun with the identification of potentially relevant standards and development guidelines. Many were found to be of too high a level for specific guidance, or had no relevance to software metrics technology. The ones which were found to most directly address metric requirements were:

a) SDS Software Policy and Management Directive No. 7;

b) DoD-STD-2167A, Defense Systems Software Development;

c) RADC-TR-85-37, Vols. 1, 2, and 3, Specification of the Software Quality Attributes.

These documents formed the core of the analysis, as they most directly addressed the development process and the associated software attributes. Integration of software metrics into the software development process was examined, including both the waterfall development method and the rapid prototyping development method. While metrics integration with the waterfall development method was relatively straightforward to define, integration with the rapid prototyping development method (itself not well understood) was not as complete.

Products and Process Attributes

The second phase of this subtask was to define the software quality and productivity factors relevant to SDS software. This identification spanned the software development life cycle from requirement definition through operation and maintenance. For the quality factors, the thirteen factors identified by RADC were examined and analyzed for applicability to SDS needs. Several were redefined or merged together, to give a new set of nine quality factors. Productivity factors were also examined, with factors relating to schedule, budget, and feedback for improvement. Primary sources for analysis were based on line of code estimates for software components, or the identification of function points. Both quality and productivity metrics are believed to be much more accurate when applied against subsets of "like" software (software domains). Accordingly, the SDS software functions were categorized and decomposed into 36 SDS software domains have distinct measurement requirements. Each domain was described through the function implemented, its level of criticality, time constraints, location, size, risk, use, and intended life cycle, as well as its quality attributes. This division served then as the basis for further measurement requirements definition.

Methodology Requirements

The third phase of this subtask was to define methodology requirements for incorporation of software measurement into the software development methodology. Four phases of methodology were defined: specification, estimation, evaluation, and tuning. The specification phase is perhaps the most difficult, requiring negotiation among developers, users, and contracting agencies to set specific requirements. The estimation phase uses the appropriate metrics to obtain predicted measures of product and process. Upon delivery, the user evaluates the actual product,

and that evaluation is compared to the metrics predictions. Finally, the metrics are adjusted as necessary to provide more accurate assessments for future developments.

Conclusions

There are several major conclusions from this subtask. The results of our review of available standards revealed that formalization of the process of estimating quality by embedding it in the development process is not well-defined. Rapid prototyping, in particular, must be directly addressed. For the waterfall development, we identified the incorporation process. We proposed a modified set of quality factors tailored to SDS requirements, reducing metric application requirements while targeting specific needs. We developed a methodology for setting quality factors based on SDS software characteristics. The SDS software was decomposed into 36 software domains for purposes of measurement application. We described the concept of software level to determine the extent of software metrics application. We identified promising productivity metrics, which appear applicable to the SDS software domains. We also developed a four phase methodology requirement of software metrics application.

Open Issues

One major open issue is the establishment of procedures for audit and review activities for the rapid prototype development process. After that is complete, then software metrics application can be integrated with it. A formal methodology for rating quality must be developed. Without that, unstable metrics models can result. Also, a formal methodology to "tune" metrics models must be defined. This includes developing formal quality rating criteria, as well as guidelines for modifying scores. Without this definition, attempts to "trade off" one quality factor against another are not likely to provide the desired results. Finally, software reuse must be aggressively encouraged. Software metrics can provide estimates of software reusability, which can be compared with actual reuse, providing a measure of one of the most promising sources of cost/schedule redirection.

0.0 INTRODUCTION

Section 1 of this document defines generic SDS software development processes which integrate software metrics technology with existing development and auditing standards. Section 2 specifies the methods for tailoring the software development process to specific software products and process attributes. Section 3 describes a methodology for defining and monitoring quality and productivity requirements for SDS software, and for establishing feedback paths for validating and tuning the metrics.

The primary objective of the metrics is to predict, throughout the development phases of the software, what the quality and the overall schedule and cost of the final product will be. In most practical cases, however, very little correlation has been consistently demonstrated between the software quality and productivity attributes as predicted by the metrics, and the real world data which are represented by the user perceived quality of the final product and the actual schedule and cost incurred. It must be realized that such weaknesses can not be easily and immediately overcome. The methodology which is proposed addresses these concerns and identifies three methods for improving the effectiveness of the current metrics. The methodology is to a great extent based on the RADC methodology which is described in "Specification of Software Quality Attributes", RADC-TR-85-37, Vols 1, 2, and 3. Several important modifications have been proposed to that methodology, which include:

a) Simplification of the existing metrics structure and redefinition of some rating assessment criteria,

b) Development of a taxonomy of software products and processes for identifying distinct software domains, and

c) Definition of user feedback paths and criteria so that the models can be continuously tuned, during the life cycle of many SDS software products.

Only those aspects of the RADC methodology that we propose modifying are discussed here. For a comprehensive treatment of the topic, see the complete set of reports referenced above.

1. IDENTIFICATION OF STANDARDS AND PROCESSES FOR SDS SOFTWARE PRODUCTS

Government and industry standards have been identified and reviewed with the purpose of defining generic development processes, including all required auditing activities, for SDS software. The metrics technology is then synergistically integrated with such processes. The metrics are intended to complement, not replace, the existing quality and productivity auditing requirements.

The two principal methodologies for SDS software development are the classical Water Fall (WF) method and the rapid prototype (RP) method. The WF method, where development proceeds through several well-defined phases, is well developed and understood and it is supported by a wealth of standards and other documentation, which include precisely defined review processes. The RP method was only recently introduced. It has generally received good acceptance, but it is not as well understood and supported by proper documentation, as is the WF method. The DoD-STD-2167A explicitly identifies these two methods as acceptable development methods for DoD software, although it does not exclude other methods.

Subsections of this section describe:

a) the standards which have been reviewed;

b) development processes which integrate quality and productivity metrics with existing auditing activities.

1.1 REVIEW OF THE RELEVANT STANDARDS

The standards which have been analyzed include:

a) SDIO Standards and Policies including: SDS software policy, Management Directive No. 7, and SDS security policy;

b) DoD Standards and policies including: DoD-STD-480, DoD-STD-2167A, DoD-STD-2168, DI-MCCR-80025A, DI-MCCR-80026A, DI-MCCR-80027A, DI-MCCR-80029A, DI-MCCR-80030A, DI-QCIC-80572, and DoD-5000.3;

c) MIL Standards and Policies including: MIL-STD-470, MIL-STD-471, MIL-STD-481, MIL-STD-490, MIL-STD-499, MIL-STD-785A, MIL-STD-882A, MIL-STD-1521, MIL-STD-1815A, MIL-STD-52779A, MIL-F-9490, MIL-Q-9858;

d) Federal Aviation Agency (FAA) standards;

e) NASA standards;

f) NATO standards;

g) Industry Standards and Policies including: IEEE, ACM;

h) System Engineering Management Guide;

i) Test and Evaluation Master Plan;

j) Current SDS software development contracts.

Most of the standards have been found to have minimal or no relevance at all to software metrics technology. These standards are not further described. The standards which have been found of relevance relative to the inclusion of software metrics within the development process of SDS software are briefly described and critically reviewed. Gaps, redundancies and other weaknesses are identified.

1.1.1 SDIO Standards

1.1.1.1 Strategic Defense System Software Policy and Management Directive No. 7

The SDS Software Policy and the associated Management Directive No. 7 is a broad outline of general guidelines and specific requirements to be applied to all SDS software systems to be developed, and in particular to the development, integration, operation and support of mission-critical software. As such, it attempts to address the requirements of a software engineering environment that would promote "software reliability, correctness, security, interoperability, portability, maintainability, and usability throughout the system life cycle." These requirements outline a framework for employing advanced software engineering practices.

The primary vehicle of the requirements is to call out specific software engineering goals and provide the implementation details in the form of references to specific government standards and publications (e.g. DoD-STD-2167A, DoD-STD-2168, etc.). In general, analysis of many of the referenced documents is performed elsewhere in this document and will not be covered here.

One referenced document that has been scrutinized for both discussion here and elsewhere in this document is the Institute for Defense Analysis (IDA) paper P-2018, "Guidelines for Tailoring DoD-STD-2167 for SDS Software Development."

Prototyping has clearly been shown to be a critical step in developing complete and accurate requirements for major (and therefore expensive) software systems. SDIO has officially supported this cost effective and timely approach to developing requirements in both the technology areas and Demonstration Validation (DemVal) phase. One of the fundamental reasons for modifying DoD-STD-2167, "Defense Systems Software Development," was to allow for prototyping to be incorporated in the system development life cycle. Unfortunately, the revised standard DoD-STD-2167A does not present a discussion or guidelines for a rapid prototyping process model. This is left to the contractor for incorporation into a particular program's Software Development Plan.

Aside from specific areas in Management Information Systems, rapid prototyping has not received the level of scrutiny and analysis given the more traditional waterfall model. The IDA paper mentioned above does present a prototyping process model for incorporation in the DoD-STD-2167 life cycle model. It is not, however, a MIL-STD or management directive.

Rapid prototyping deserves a more in depth treatment and, in particular, development of a Government approved standard of requirements. Measurement requirements should be considered for inclusion in this standard. Further discussion of the integration of software metrics in the rapid prototyping development process is covered in section 1.2.3.

Other than the referenced documents, specific requirements of the SDS Policy and Management Directive call out additional procedures to be followed during development and the desired goals to be achieved. The requirements generally do not specify individual measurement

requirements to be levied against the software development process or resulting products. Some of the requirements do, however, clearly indicate that measurement requirements should be, or in some instances, must be developed in order to accurately assess conformance. In other requirements, measurement requirements could be levied to assist in achieving the stated goals.

In particular, the following requirements (as stated in Section D of the Management Directive) were considered for derivation of specific measurement requirements.

Requirement 2, Prototyping, states that an incremental prototyping approach should be encouraged in order to facilitate early detection of errors, quick validation of requirements and timely determination of feasibility. The principle of "build a little-test a little-learn a lot" is emphasized. In an attempt to "learn a lot," measurement requirements levied against the iterative phases of the prototyping model (as detailed in IDA Paper P-201, "Guidelines for Tailoring DoD-STD-2167 for SDS Software Development"), measurements of incremental size and complexity as well as reliability and performance could provide early insights into determination of feasibility.

Requirement 3, Supportability, calls for a plan to address support for both operational and support software, procedures used for software operation and maintenance, growth patterns to accommodate lessons learned and technological advances, among other items. This plan must also include specific supportability criteria used to evaluate software to assess compliance with supportability requirements. Specific measurements dealing with usability, interoperability, maintainability, etc. could be incorporated as part of the supportability criteria.

Requirement 4, Risk Reduction, calls for several actions to be incorporated in the software development and program management plans in order to minimize the risk associated with a development effort. In particular, this requirement calls for a software failure mode, effects and criticality analysis such that the appropriate actions can be taken to ensure a successful software life cycle. Specific software reliability measurement requirements should be levied against each of the life cycle phases.

Requirement 9, Portability, outlines specific goals that must be achieved with respect to the ability to port delivered software across computational environments. Measurement

requirements levied against early phases of development therefore may assist in avoiding potential redesign and recoding late in the development life cycle.

Requirement 10, Testing, calls for a testing strategy for detecting errors in requirements, designs, and code in addition to identifying the need for metrics to predict, estimate, and evaluate critical software quality attributes at each software development stage. To some degree, the above requirement could be viewed as a high level measurement requirement. However, it allows the contractor to select the measurements to be applied. More specific measurements could be specified to provide for a more uniform application of metrics and provide for a more meaningful interpretation of the results (with respect to similar development efforts).

Requirement 11, Reuse, calls for the establishment of a controlled repository for software in order to promote reuse. Contractors must evaluate software available for reuse including a review of the values for pertinent software quality factors. In addition, RFPs will identify components to be developed that may be included in the repository. Past history has shown a reluctance towards reuse of components developed outside of the development environment where it could be used. This is due in part to the risk incurred in using components whose quality attributes are unknown or unreliable. There is a clear need to establish uniform measurement requirements to be applied against components being considered for inclusion in a code repository or reuse library.

1.1.1.2 SDS Security Policy

This policy, as stated in CDRL J001 for TASK 6.6 of the SDI System Architecture and Key Tradeoff Study, Phase IIC, under contract number MDA 903-85-C-0064, outlines the rules and guidelines for the development and maintenance of information systems' security. This document sets forth thirty five specific policy statements which cover computer security, communications security and those aspects of physical, administrative, personnel, operations and emanations security that pertain to the protection of information within the system.

These policy statements address the high level needs for security through the SDS Development Phases (Research, DemVal, Full Scale Development (FSD), Production/Development, and Operations and Maintenance). However, they do not directly levy specific requirements against specific SDS components. Their primary use would be to assist in

deriving security requirements for a specific program. Indirectly, policy statement thirteen calls for the enforcement of life cycle assurance provisions and policy statement seventeen calls for security relevant and mission critical elements to be verified. In addition, policy statement twenty four calls for secure, fail-safe, and fault-tolerant information systems to be used during launch of SDS components.

The above statements, however, do not provide direct metric requirements against specific software components. They can be used as general guides in selecting the appropriate measurements.

1.1.2 DOD Standards

Only a few DoD standards have been found to have any significant impact relative to the application of metrics to the software development environment. Those standards are discussed in the following paragraphs of this document.

1.1.2.1 DoD-STD-480B, Configuration Control

This standard establishes the requirements, formats and procedures for maintaining and documenting configuration control of all configuration items (software and hardware), and for controlling the Engineering Changes Proposals, Deviations and Waivers, Revisions and Changes. The standard does not have direct implications on Software Quality or Productivity.

1.1.2.2 DoD-STD-2167A, Defense System Software Development

The DoD-STD-2167A, Defense System Software Development, is the standard most relevant to software metrics. DoD-STD-2167A establishes the requirements for software development which are applicable throughout the system life cycle. The standard was updated in Feb. 1988. The major motivating issues for updating the standard were the following:

a) Removal of the restrictions on development methodologies. The new version explicitly states that the contractor is responsible for selecting development methods, like the RP. It removes top-down, and the sequential development phasing implications, as the default methodology;

b) Removal of excessive "How To" requirements like the use of structured analysis tools, top-down design methodology, Program Design Languages, etc.;

c) Encouragement of development and use of reusable software;

d) Removal of language specific terminology so that incompatibility with Ada are removed.

The DoD-STD-2167A requires the contractor to specify a software development process which supports all the formal reviews and audits which the contract requires, and which must include the following major phases:

- System Requirements Analysis/Design
- Software Requirements Analysis
- Preliminary Design; Detailed Design
- Coding and CSU Testing
- CSC Integration and Testing
- CSCI Testing
- System Integration and Testing

These System Development Phases are shown in Fig. 1-1, Software Acquisition Quality Metric Functions. Such activities may overlap, and may apply iteratively or recursively. The DoD-STD-2167A requires that reviews and audits be consistent with the guidelines provided in MIL-STD-1521B, which is discussed later in this document.

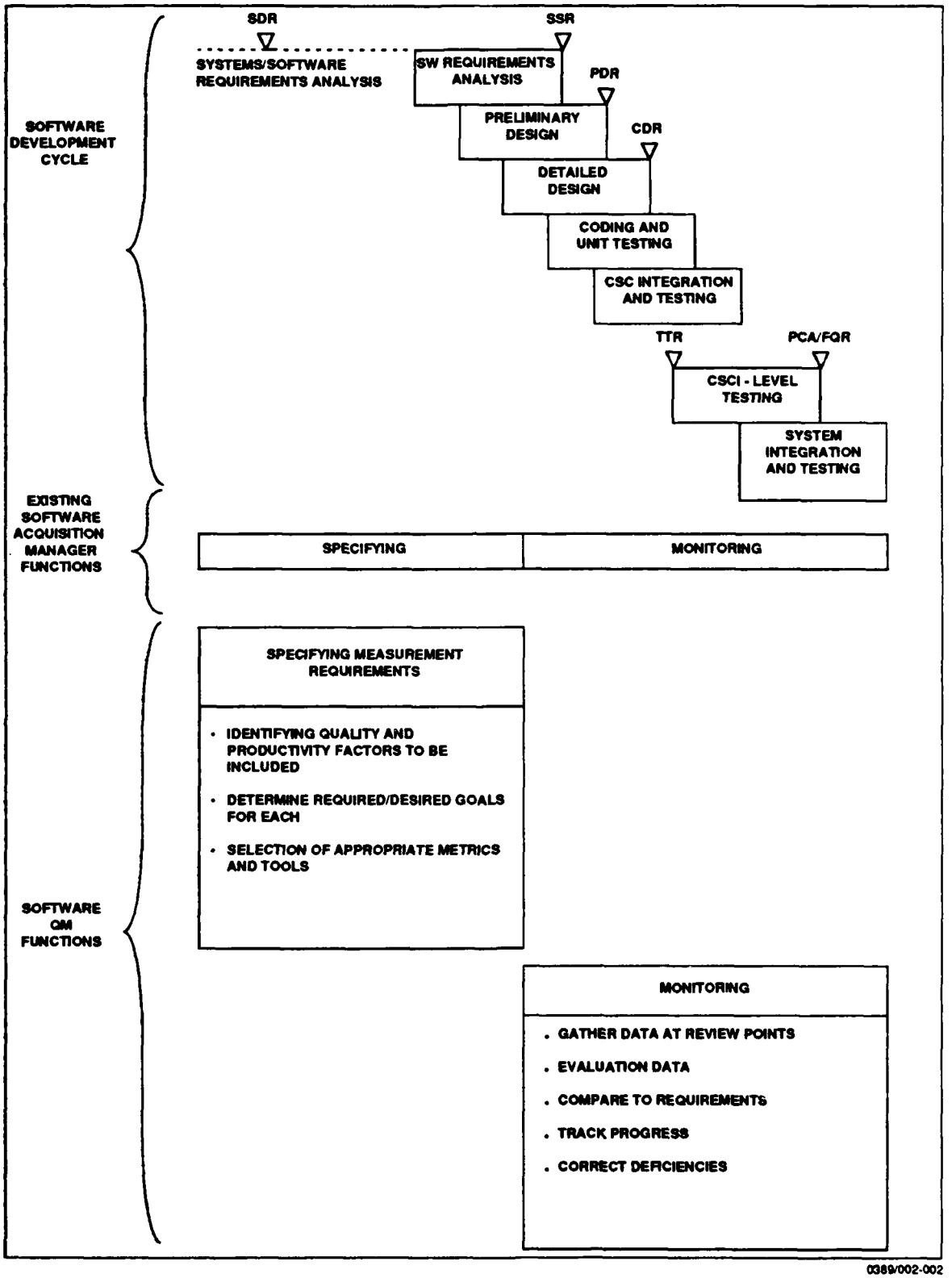


Figure 1-1 Software Acquisition Quality Metric Functions

Figure 1-1 refers to the WF development model, and it defines reviews at the end of all major development phases. Review activities include:

a) System Requirement Review (SRR), at the completion of the System Requirement Analysis;

b) System Design Review (SDR), at the completion of the system design. these first two phases are often considered part of the system, rather than software, life cycle;

c) Software Specification Review (SSR), at the completion of the software requirements analysis. SSR is generally considered the first review within the software life cycle;

d) Preliminary Design Review (PDR), at the completion of the software preliminary design;

e) Critical Design Review (CDR), at the completion of the software detail design;

f) Test Readiness Review (TRR) at the completion of the CSC integration and testing;

g) Functional Configuration Audit (FCA), at the completion of the CSCI testing;

h) Physical Configuration Audit (PCA) and the Formal Qualification Review (FQR), at the completion of the system integration and testing. FQR is considered outside the software life cycle.

The DoD-STD-2167A defines the high level activities which must be conducted in each development phase; examples are: the software development plan, the interface requirements specifications, etc. It also defines a common set of evaluation criteria which apply to all items to be evaluated. Typically a subset only of the criteria apply to each item. The evaluation criteria are the following:

a) Internal Consistency;

b) Understandability;

c) Traceability to the indicated documents;

d) Consistency with the indicated documents;

e) Appropriate analysis, design, or coding techniques used;

f) Appropriate allocation of sizing and timing resources;

g) Adequate test coverage of the requirements.

Many of these criteria are included among the measurands which contribute to the scoring of software metrics. The example also indicates the role of the Software Measurement Process in the timeline of the system acquisition process.

1.1.2.3 DoD-STD-2168, Software Quality Evaluation

The standard describes the requirements for the development, documentation, and implementation of a software quality program, including conducting evaluations of the quality of the software. The Data Item Description (DID) applicable to this standard is DI-QCIC-80572. As with DoD-STD-2167A, this standard is designed to be tailored for each contract. The standard does not address the specific quality requirements which must be addressed and demonstrated during the development process. Such requirements are addressed in MIL-STD-1521B.

1.1.2.4 DI-QCIC-80572, Software Quality Program Plan

This Data Item Description, Software Quality Program Plan, identifies the procedures to be used by the contractor to perform and document the activities related to the Software Quality Program specified in DoD-STD-2168. The DID, however, describes those procedures at so high a level that it does not affect software metrics.

1.1.3 MIL Standards and Policies

1.1.3.1 MIL-STD-1521B, Reviews and Audits

This standard prescribes the requirements for the conduct of technical reviews and audits on Systems, Equipments, and Computer Software. They include: SRR, SDR, SSR, PDR, CDR, Test Readiness Review (TRR), Functional Configuration Audit (FCA), Physical Configuration Audit (PCA), Formal Qualification Review (FQR), and Production Readiness Review (PRR). This standard must be tailored to the specific contract requirements. The software quality factors' requirements of each CSCI are reviewed during the SSR. The software quality factors are not discussed in any detail, however.

1.1.3.2 MIL-STD-1815A, ADA Programming Language Reference Manual

This standard establishes the specifications of the Ada language. Compatibility with Ada terminology and structures was one of the motivators for updating the old DoD-STD-2167. No significant direct impact on software metrics requirements have been found as a consequence of the identification of Ada as the standard SDS software language.

1.1.3.3 MIL-Q-9858, Quality Program Requirements

This standard, Quality Program Requirements, refers to the organizational requirements of a Quality Program, and it specifically addresses hardware considerations. It does not impact software quality metrics.

1.1.4 Federal Aviation Agency (FAA) Standards

The FAA is the Government agency with the responsibility of certifying the airworthiness of civil aircraft. The FAA issues Federal Air Regulations (FAR) which establish broad guidelines for the development, verification and certification processes. Details are covered in Advisory Circulars (AC) which in turn reference specific documents which are typically generated by industry consortiums. One of these documents with considerable relevance to this study is the Radio Technical Commission for Aeronautics (RTCA) Document No. RTCA/DO-178A, March. 1985, titled "Software Considerations in Airborne Systems and Equipment Certification". The document is referenced by AC 25.1309-1 (Airplane System Design Analysis), and deals with the development, verification and validation process of flight software. The document has provided the guidelines for the certification of the latest generation of aircraft with digital flight control systems.

During the preparation of the document, techniques were examined for estimating the post verification probabilities of software errors and therefore of the resulting operational failure rates. The objective was to develop numerical targets for maximum failure rate as a function of the criticality of the system. The conclusion reached, however, was that the current level of technology could not develop those estimates with the required level of confidence. As a result the document did not address numerically expressed reliability targets.

The document outlines a software development process which is consistent with the WF model described in the DoD-STD-2167A. The DO-178A model, similar to the DoD-STD-2167A model, needs to be tailored to the specific requirements of the application, specifically its criticality. For this purpose, three levels of criticality were identified: critical, essential and non-essential. The corresponding software was identified as being of Level 1 (the most critical), Level 2, or Level 3. The determination of the criticality is negotiated during the earliest phases of system definition, among all parties involved, including the certification agency, the avionics manufacturer and the airframer. The implications of the criticality classification are extremely important because the development effort required, and therefore the development cost, is much higher for more critical software than it is for less critical software.

It is very cost effective to apply the concept of software levels to SDS software. By doing so, the development of software which is not critical can be scaled down, while the development of more critical software requires correspondingly more formal, stringent and elaborate, and therefore more costly, development strategies. The following must be defined:

a) The levels for SDS software, as a function of the criticality and possibly of other factors such as size, complexity and cost;

b) A methodology for establishing the level of criticality of a software product;

c) The level of development, auditing and reviewing effort, including software metrics, which is appropriate for each software level.

1.1.5 NASA Standards

The only NASA-wide standard on software development was developed in 1976, and is applicable to mission critical flight software. The standard is very high level (six pages) and seldom used. Currently, each NASA project publishes its own standards, which typically are quite good and reflect current accepted software engineering practices. As an example, NASA/Goddard has developed its own standards for the development of ground based software. The NASA Space

Station program has also developed a set of standards for software development. They are not yet available for review.

1.1.6 NATO Standards

NATO has not published a standard for software development. A document is soon to be published by the Working Group #8 of the Advisory Group for Aerospace Research and Development (AGARD) relative to software development. It includes a description of the software development process which is consistent with the WF model of the DoD-STD-2167A (although a brief reference to the RP model is also made), the supporting methodologies, and the required documentation. Reference is made to DOD-STD-2168 relative to quality requirements, but no reference is made to metrics technology. No NATO standard or publication was identified which has significant effects on metrics technology.

1.2 INTEGRATION OF SOFTWARE METRICS IN THE SOFTWARE DEVELOPMENT PROCESS

The two processes which are best suited for SDS software development are the WF and RP models. Other development process models exist, in addition to the WF and RP. They are all, however, derivatives and/or combinations of these two primary models. An example is the Spiral Model (SM) which is an iterative development process which may include both the WF and RP models for supporting different development phases of system components depending on the estimated technology risks. The inclusion of a Risk Analysis (RA) in the development process is an important component of the SM. High risk items need to be resolved at the earliest phases of development to prevent costly bottlenecks in the later phases. The SM model can be adjusted for accommodating the development of products with different technology risks; RA can and must also be included in the WF and RP models. [IDA PQ018]

The auditing, review and software metrics requirements of only the RP and WF models are discussed in this document. The corresponding requirements of other development processes can be developed from these two requirements.

1.2.1 Application of Metrics to Software Development Processes

An extremely important and traditionally overlooked requirement of metrics is the need for meaningful results early in the development process. Statistics gathered on software development for several complex weapon systems have shown that approximately 70% of all "software errors" actually occur early in the development cycle, but are not detected soon enough, and propagate into software production, deployment, and operation. The impact of this delay is typified by enormous overruns, large slips in schedules, and systems delivered with less than the required capability. Some of the causes of this situation are improper flow-down of requirements to design to code, inadequate integration of modifications during development, and incomplete development testing. Correction of software problems often cost more than the development cost of the original software involved because of extensive de-bugging, replanning, redevelopment and revalidation.

An important consideration, then, is to make sure that metrics are identified that give early insight into requirements generation, preliminary design, detailed design, test requirements, interface requirements, and so on. This is especially true for the Waterfall development, since code is not developed until significant amounts of development funds have been spent. Metrics that have been shown to have moderate to high precision and accuracy, such as McCall and Matsumott, various linear regression techniques, and analyzers such as McCabe's Cyclic, have little to moderate use during the requirements phase when assessment is most needed. Careful attention to this problem leads to selection of other metrics that can be used at the top of the Waterfall. Examples are the extraction of metric data from requirements generation tools, such as Technology for the Automated Generation of Systems (TAGS), and even from formal document review results.

In the case of rapid prototyping, the inherent advantage exists of early development of code, implementation of interfaces, and testing, so that quantitative measures can be utilized early enough to indicate how to avoid problems rather than where problems exist and need to be fixed. Many metric techniques should lend themselves to Initial and Advanced Prototypes, as well as Full-Scale Development. Techniques should also be defined to address Concept Formulation phase as well.

1.2.2 Integration of Software Metrics in the WF Development Process

The DoD-STD-2167A, as previously stated, does not enforce the WF development method as a standard, but also encourages other development models, like the RP. The entire methodology described in the DoD-STD-2167A, however, is consistent with the specify, then design, then build, then test, document driven, WF model. The WF model progresses from one phase to another in an orderly and predictable fashion. The review and audit activities are then scheduled at the end of each phase, according to the scheme described in section 1.1.2.2. Software metrics review activities should occur concurrently with the review and audit activities. Software metrics review can be effectively integrated with the auditing reviews as they are outlined in the DoD-STD-2167A, at least relative to the WF development processes. The scores of each metric can be reviewed as a part of, and concurrently with, the corresponding audit activities. This approach minimizes the cost of the review process, prevents duplication of tasks, and provides a more complete overall assessment of progress, which neither the audit activities alone, nor the metrics, could otherwise provide. Furthermore, the approach provides some feedback paths early in the development process, to tune the metrics, by measuring the correlation between metrics scores and the outcome of the audit activities specified in the DoD-STD-2167A. These early feedback paths, and most importantly the paths at the end of the development phases, are important elements of the proposed methodology, and are further discussed in later sections of this document.

1.2.3 Integration of Software Metrics in the RP Development Process

A development approach based on the RP model has the promise to be very effective, at least in some applications. The specification of the requirements of a large system is a most complex and challenging task, which until recently has not been supported by well developed methodologies. Direction for Software Development using an RP process is provided in the draft Guidelines for Tailoring DOD-STD-2167A for SDS Software Development. The RP model, as defined in the guidelines, provides the user with the capability of interacting with embryonic implementations of the requirements, very early during the development cycle, so that any inconsistency, incompleteness, or misinterpretation of the requirements can be identified and corrected. A prototype is built after a minimum effort is spent for developing the requirements and the design specifications. The RP approach is best suited for resolving high risk items, like communications

within a distributed architecture, and complex user interfaces. The prototype implementation must be very flexible so that it can be easily modified.

The RP process may either produce a throw-away model which only supports the early phases of the development process (the acronym of that process is TA-RP) or may produce a model which is evolutionary during the entire development process (E-RP). In the latter case, each successive design and implementation version includes an increasing number of functional specifications and design details. In either case, the process benefits from the active participation of the User who has the opportunity to gain a better understanding of the requirements, and to communicate them directly to the system designers. The interaction with the User in these early development phases, is a most noteworthy characteristic of the RP process.

The TA-RP process implies the development of a prototype version of the system for the purpose of refining the User requirements and/or resolving high risk, critical design specifications. The useful life of the prototype ends when the objectives have been achieved. During its useful life the prototype supports the System Requirements Analysis/Design and, possibly, the Preliminary Design, limited, of course, to the scope of the prototype. The prototype must then support the corresponding audit and review activities, including SRR, SDR, SSR and PDR. Software metrics must be concurrent to, and synergistic with, such activities. The remaining development phases are supported by the WF model, and the corresponding audit and review activities, and the software metrics are consistent with that model.

The E-RP model is a continuous, iterative process which is repeated several times. Every new cycle implies the development of a prototype which satisfies an increasingly refined set of requirements, and enhanced design and implementation.

The insertion of the metrics technology within an RP driven development process is a difficult task because little or no guidance is available. The RP model is more recent and less understood than the WF model, and is not supported by clear and precise guidelines relative to activities and documentation requirements. The establishment of review and audit activities for the E-RP process presents unique challenges. For example, the phasing process from the RP to the WF model, as the specifications get more and more complete and understood, is not adequately described in any standard, military or otherwise. The resulting difficulties apply not only to the

application of metrics technology, but also to the definition of products, of review and audit activities, and of documentation requirements.

Recently, the SDS System Engineer (SE) has been tasked with developing a prototyping model and methodology along with associated standards and guidelines. These guidelines are to be incorporated into a Computer Resource Life-Cycle Management document due for external review in the June 1989 time frame. They intend to follow up with a white paper on prototyping methodology in mid to late summer 1989. It is unclear, at this point, what will be included in this report.

In a technical interchange meeting, the SE presented a preliminary prototyping model. The model is currently only described at a very high level with minimal detail regarding specific application of the model and associated requirements.

The approach of the SE prototyping model emphasizes an incremental development via iterative refinement and includes formal control mechanisms. Each iteration results in a specified degree of functionality (baseline). Although this closely parallels the Evolutionary Rapid Prototyping (E-RP) model presented earlier, the primary goal of the SE model is correctness of requirements. Once the requirements have been completed, the prototypes are discarded and a formal software development cycle begins (they did feel that some code, at the lower levels, could be reused). The E-RP model discussed earlier assumes that, depending on the level of review and audit activities employed during the development, some significant portions of the prototype could be migrated into the production development. The SE asserts that the role of prototyping ends with the completion of the DemVal phase.

The SE model is very similar to a spiral model of S/W development where each iteration follows four steps:

1) Determine objectives/constraints

2) Experiment/demonstrate/test

3) Assess risk/evaluate alternatives

4) Refine/define specifications

Each iteration results in an incremental build (P1,P2,...Pn) of the software and the associated requirements. This process continues until "some point" at which full scale development may begin. The SE did not quantitatively define the "point" at which the prototyping is complete, nor did they present a methodology for measurement of progress through iterations.

The SE model proposes four classes of prototypes:

1) Proof of Concept - used to prove technical feasibility of a new technology in order to reduce risk

2) Requirements Definition/Clarification - used to define/clarify requirements that are unknown, vague or unclear

3) Design/Implementation Refinement - used where requirements are known but implementation method/technique is uncertain or alternate approaches exist

4) Product Approximation - used as a model of the final software product

The SE has provided a proposed mapping of the four classes of prototypes to the recommended required software standards (see Figure 1-2).

DATA ITEMS	PROTOTYPE CLASS 1	2	3	4	COMMENTS
SOFTWARE DESIGN DOCUMENT (SDD)	X	X	X	X	TOP LEVEL FOR 1 & 2
VERSION DESCRIPTION DOCUMENT (VDD)			X	X	
SOFTWARE TEST PLLAN (STP)				X	
SOFTWARE TEST DESCRIPTION (STD)				X	
SOFTWARE TEST REPORT (STR)	X	X	X	X	PROTOTYPE REPORT FOR 1,2 &3
SOFTWARE PROGRAMMERS MANUAL (SPM)				X	
SOFTWARE REQUIREMENTS SPECIFICATION (SRS)	X	X	X	X	
INTERFACE REQUIREMENTS SPECIFICATION (IRS)		X	X	X	
INTERFACE DESIGN DOCUMENT (IDD)			X	X	
SOFTWARE DEVELOPMENT FOLDER (SDF)	X	X	X	X	ENGINEERING NOTEBOOK FOR 1, 2, & 3
ITEM TOTAL	(4)	(5)	(7)	(10)	

CLASS 1 = PROOF OF CONCEPT
CLASS 2 = REQUIREMENTS DEFINITION
CLASS 3 = DESIGN/IMPLEMENTATION REFINEMENT
CLASS 4 = PRODUCT APPROXIMATION

0389/002-003

Figure 1-2 Required Software Standards by Prototype Class

The SE did not provide definitions or proposed tailoring for the required documents (however, the SE indicated this would be forthcoming in the Computer Resource Life Cycle Management document). They also did not propose any recommended software measurements to be used during the prototyping process.

It is clear that in order for this proposed prototyping model to fulfill its stated objective of "reduction of risk and cost" it must include a specific measurement approach as part of the iterative development methodology, without which selection of specific metrics and their associated insertion points into the development process would be very difficult and probably would be of limited value.

2. PRODUCTS AND PROCESS ATTRIBUTES

Section 1.2.2 discusses a generic methodology for inserting metrics technology in the SDS software development process. That methodology must be tailored to the specific characteristics of each product (software typing) and of each development process (process typing). A most critical step of this process is to assign the proper software quality attributes to each software product. We started this study by analyzing the suitability of using the hierarchical framework of quality attributes developed for RADC, and which is also referenced in the SDS TEMP document. During our study we found that the RADC framework is excellent. We identified, however, several areas of improvement which are consistent with the pragmatic objectives of this project. The proposed changes decrease the total number of quality attributes from thirteen to nine. The definitions of reliability and efficiency have been considerably modified to better reflect the user concerns in that areas. New measurands must be developed which are consistent with the new definitions.

2.1 SOFTWARE QUALITY FACTORS

The set of software quality factors spans the life cycle from requirements definition through development and test and the maintenance and modification of operational software. In the ***requirements phase***, software quality is assured by 1) verifying traceability from system requirements to software requirements, 2) expansion of software specific performance requirements such as port-to-port timing and accuracy and operational requirements such as reliability, availability and survivability, and 3) allocation of these requirements to CSCIs, CSCs and CSUs appropriate to the functional hierarchy and the nature of the software process. In the ***development phase***, software quality is enhanced by the use of a user-friendly software development environment that provides developer and tester access to data required for their function including appropriate requirements, data interfaces for each task whether it is development of CSUs, Unit testing, integration into CSCs/CSCIs and testing. The software development environment should:

a) enforce development standards;

b) provide metrics for managing development progress;

c) provide configuration management throughout the multi-user development process;

d) automate traceability and documentation to the maximum extent possible, and

e) provide a re-use library.

The development phase also involves the use of simulations/emulations, including hardware/software in the loop, for verifying that the developed code meets the software requirements and validating that system requirements are met. The development process culminates in the successful initial operational capability of the host system. In the ***operational phase***, software functions include: modifications, upgrading and maintenance. Related software quality factors are:

a) change requirements;

b) software documentation;

c) software modularity;

d) field-site testability and diagnostics; and

e) verification and validation.

The ability to achieve software quality in the operational phase is built into the system during the requirements and development phases. The implementation of operational phase functions is accomplished through the hardware and software provided for this function and the level of proficiency attained by proper selection and training of personnel.

Considerable work toward defining software quality factors has been accomplished by RADC. This section describes the RADC software quality framework and their quality factors. The RADC quality factors address three areas of user concerns:

a) operational performance,

b) design, and

c) adaptation.

The design area addresses correctness (a measure of the extent to which the design conforms to specifications and standards; not correctness of the software in the sense of being fault free) and maintenance and verification. Adaptation concerns address the issues of the ease of the

software to operate in different environments, to be flexible, easy to change and reuse. These issues are further expanded in the following sections of this report.

In the following paragraphs we discuss the RADC quality factors and our proposed modifications to improve their applicability to SDS software.

2.1.1 Performance

Performance issues address the capability of the SDS software of producing correct results within the allocated time. The quality factors included in software acquisition are:

a) efficiency;

b) integrity;

c) reliability;

d) survivability; and

e) usability.

Usability is a design consideration, so we propose to move this criteria from the performance area to the design area.

The best interpretation of the meaning of each quality factor is the proposed rating factor. The rating factor is the software product attribute which the metrics attempts to estimate during the software development process. The rating factor must reflect the user concerns relative to the specific issue under consideration. During this analysis, however, we found that some rating factors did not address the core of the user concerns in some specific areas, and therefore we modified the definition of the quality factor by changing the quality rating criteria.

2.1.1.1 Reliability

The RADC quality rating for this quality factor is expressed in terms of number of software errors per 1000 source lines of code (SLOC), occurring during a specified software development phase. The corresponding quality metrics and metrics elements are based on:

a) testing if the requirements for accuracy have been appropriately included;

b) the simplicity of the design and of the implementation; and

c) the robustness of the software (i.e., the capability of the software handling off-nominal situations).

We have the fundamental concern that neither the quality rating, nor the metrics elements address the primary user concern in the area of software reliability.

For this report, software reliability is defined as the probability that the software will not cause the failure of a system for a specified time under specified conditions. This definition is consistent with the IEEE standard glossary of Software Engineering Terminology definition for software reliability. The parameter of reliability is failure rate, defined as the ratio of number of failures to a given unit of measure, for example failures per unit time.

We propose that the definition of reliability be consistent with the IEEE standard definition. The corresponding quality factor rating criteria must be based on measured reliability or failure rates. Changing the metrics elements is a challenging task which is not well supported by the current technology. We propose to initiate an effort in this area which has the very worthwhile objective of analyzing and developing the technology for predicting operational failure rates of critical software.

2.1.1.2 Survivability

The RADC rating factor of survivability is again expressed in terms of number of errors per SLOC. The only errors considered, however, are the subsets of total errors which affect survivability. The IEEE standard definition of survivability is the built-in capability of providing continued execution in the presence of a limited number of faults. In other words survivability is the capability of the software to perform the required functions when a portion of the system is inoperable. This definition directly addresses the user concerns in this area. The rating of this quality factor can be based on measures of:

a) the fault detection capability or coverage (percent of total possible faults which the system is capable of detecting); and

b) the reconfiguration capabilities expressed as the percentage of the software which is at different levels of fault tolerance.

By combining these two measures, the probability that the software will perform in the presence of faults can be assessed.

2.1.1.3 Integrity

The same considerations previously made for reliability and survivability (errors per SLOC, subset of total errors affecting quality factor) also apply to integrity. For sake of conciseness, they are not repeated here. The issue of integrity is complex, and there is the possibility that metrics may not be an appropriate technique for providing accurate estimate of such critical requirements. Integrity performance requirements and evaluation criteria are described in DoD-STD-5200.28, which may far exceed what can be reasonably expected from metrics, in this very complex area. This consideration is made for the purpose of analyzing the possibility of removing integrity from the proposed quality factors. We have not made a firm decision on this matter, at the present time.

2.1.1.4 Efficiency

RADC metrics address the user concern for high throughput and minimal wastage of computing resources by means of the quality factor efficiency. Efficiency refers to the effective use of resources. The rating of efficiency is accordingly defined as the ratio between actual resources utilization and allocated resources. The RADC metric elements address effectiveness of:

a) processing;

b) data usage;

c) data storage.

There are two aspects of this quality parameter that cause it to be incomplete, as it is defined. First, the proposed rating is dependent on the estimate of the allocated resources. A generous allocation can make a poor design and implementation look good relative to a good design. Second, the user is generally concerned more with performance efficiency, like computational or communication throughput, than resource utilization. However, the requirements for efficiency must be included, because much of the SDS software will execute in environments

where weight, size, and power allocations are tightly constrained. Efficiency requirements must be applied very judiciously, however, to minimize the undesirable impact on cost and schedule.

There are several levels of efficiency requirements. At the low end, some minimum efficiency requirements are clearly necessary, easy to achieve, and must be required to prevent designs which needlessly waste resources. Such requirements must always be stated, implicitly or explicitly. At the other end of the spectrum, maximum efficiency requirements must be established for those cases where a software function can only be performed if very stringent efficiency requirements are satisfied. In such cases, nothing is more important than efficiency. All other quality requirements, even those which are in negative correlation with efficiency, must be set to a level which is not in conflict with the high efficiency requirement. Only a very small fraction, if any, of SDS software is expected to have such efficiency requirements. Such a degree of efficient use of the resources almost never needs to be a critical requirement of an entire software system. It has been demonstrated many times, and often very painfully, however, that the price paid for "super efficient" code can be very high as proven by the fact that that criterion is in negative correlation with almost all other quality factors, even within the RADC framework. Previous studies have shown that 90 percent of execution time is spent executing within approximately ten percent of the code. Putnam Texel, a recognized Ada development expert, further asserts that typically only three percent of the Ada source code of a large system is a candidate for a very high level of performance optimization. Efficiency targets set for the software as a whole can be extremely detrimental to the total development effort. An additional consideration to make is that the rapid advances of hardware technology which we are experiencing puts less premium on efficient code. SDIO is clearly aware of the negative impact efficiency requirements may have with respect to other necessary requirements, and it is stated in the last revision of the SDS TEMP document that hardware limitations or constraints will not drive the software requirements.

In between the extremes of only minimal efficiency requirements, and overriding efficiency requirements, is that applicable to most of the SDS software. For that software, efficiency requirements are important, but not so important that all other quality requirements can be overridden by it. In these cases, a difficult negotiation must be made among the requirements for efficiency, and the requirements for all other factors which are negatively correlated with efficiency. This is an area requiring further research, as discussed in section 3.1.

The recommendation is to augment the "efficiency" quality factor as it is currently stated in the RADC metrics context. An additional quality factor "throughput" is proposed which addresses the user concerns of computational and communication throughputs. A rating based on the comparison between throughput requirements and demonstrated throughput capabilities is suggested. The development of appropriate quality metrics elements for this new quality factor, is beyond the scope of this study.

2.1.2 Design and Adaptation

In the RADC metrics framework a distinction is made between user concerns related to design validity like maintainability, verifiability, etc.; and those related to the adaptability of the product like expandability, flexibility, interoperability, etc. The distinction is not justified because adaptation concerns are themselves much related to the design, as previously discussed. The major problem, however, with the quality factors in both categories as currently defined, is that many of the "ilities" are overlapping and redundant. For example, a product which is easy to maintain, must also be easy to verify; the converse must also be true. Both factors are estimated by the same set of criteria: modularity, self-descriptiveness, simplicity, and visibility. Maintainability also uses the consistency criterion, which is the only difference between the two criteria sets. Two metrics apply to the factor consistency: procedure consistency and data consistency. Procedure consistency is aimed at estimating the use of standard procedures for control flow and data flow representation, calling sequences and I/O protocols, error handling, and naming conventions. Data consistency is aimed at estimating the use of standard procedures for naming conventions, data representation, and database structures. The two metrics are applicable to verifiability because if standard procedures and standard data conventions are throughout the design, then the software is easier to verify than in the case where several types of procedures and data representation must be verified. In the former case only standard procedure and data representation must be verified. In the latter case each procedure and representation, and the compatibility among them, must be verified. The conclusion of overlapping and redundant factors is further substantiated by the fact that considerable overlap also exists among the metrics elements which support the estimate of the remaining quality factors. We propose to reduce the number of quality factors to a smaller number than in the RADC metrics, by grouping together several highly related quality factors. The advantages of such approach include:

The quality factor reuse is a new factor which includes:

a) the old Reusability factor, which is a measure of the ease of reuse of an existing product, or elements of that product; and

b) a new factor which evaluates the amount of reused products in the new product being evaluated.

This new definition is important because it may be used to encourage software reuse, probably the most effective single tool available for improving software productivity. The development of the metrics elements which support the new structure of software quality factors and the new factors is beyond the scope of this study.

2.2 SDS SOFTWARE

Twelve major SDS software functions were broken down into thirty-six software components. This decomposition does not intend to represent earlier functional decomposition into subfunctions and sub-subfunctions; rather, it is an attempt to break down the software into components that pose distinct requirements for quality metrics.

2.2.1 SDS Software Structure

In this section the different types of SDS software are identified using a variety of documents like "SDS System Level Functional Analysis and Requirement Definition" and "TEMP." SDS software falls into one of the following classifications:

a) Support software elements, such as development and verification environment and tools (compilers, static and dynamic analyzers, operating systems, run-time support, communications, etc.).

b) Simulations and emulations (scenarios, technology, etc.).

c) Management tools (configuration management, metrics, etc.).

d) Operational software (space based, ground based, training, communications, etc.).

The decomposition structure is developed to the point that each software program in the lowest tier has uniform characteristics so that it entirely fits in a single software domain. All identified SDS software elements are then classified according to the software domain in which they fit. Other decomposition schemes are also appropriate for other purposes. A categorization scheme based on cost categories has been previously used. Similarly, the decomposition and categorization to be used by the SE is expected to be different from what is developed here, although it is not yet available. However, the intent here is to differentiate SDS software based on similarity of characteristics and functions, then to aggregate based on similarity of quality requirements. This provides relatively homogeneous sets of software domains, which is expected to improve metrics correlations.

2.2.2 Decomposition of Major SDS Functions

In order to develop software quality domains, the major SDS functions were first composed to a lower level of detail. The resulting categories of major SDS functions to be considered are listed below:

a) Detect

b) Identify

c) Track

d) Communicate

e) Assess Situation

f) Assign and Control Weapons

g) Guide and Control Weapons

h) Control Platforms

i) Simulate

j) Support Development

k) Support Acquisition

l) Support Management

2.2.2.1 Detect

Determine the presence of threat objects and potential threat objects. For SDS, detection is performed by several system elements of differing sensor types, operating singly or in groups, and having different basing and timelines. Detection software is broken down into the following components. All are critical components, but differ somewhat in characteristics. For instance, real-time processing to detect cold bodies at long range involves higher Technology Risk than does detection of plumes.

a. Detect Plumes - Use of short-wave infrared (SWIR) optical sensors to detect plumes of boosters and Post-Boost Vehicles. Because of the limited viewing time available, and the long range required of these sensors, much quicker response will be required for detection processing than for the other components of this function.

b. Detect Cold Bodies - Use of long-wave infrared (LWIR) optical sensors to detect reentry vehicles. Although this component also involves processing of optical sensor detections, the process is much different; it is performed over many more targets, but does not involve processing against a cluttered background. Basing requirements also vary.

c. Radio Frequency (RF) Detect - Use of ground-based radars to detect boosters, post-boost vehicles (PBVs) and reentry vehicles (RVs). This software component has differences in basing and in the type and amount of processing that must be performed.

2.2.2.2 Identify

Differentiate threats from non-threats. Several components are involved in separating these two classes.

a. Resolve Objects - To a large extent this subfunction is performed by sensor hardware, but a software component is involved when stereo viewing is required from multiple passive sensor platforms, where computations may be performed onboard spaceborne platforms.

b. Discriminate - This software component must fuse a priori information, learned information, and measurements to identify threatening objects. Primarily a Battle Management

software function, this software differs from object resolution software in basing, timeline, and the class of algorithms that may be employed (parallel, artificial intelligence (AI), etc.).

c. Assess Kills - This software component must operate on data from sensors and weapons to predict and assess kill event parameters.

2.2.2.3 Track

This SDS function keeps track of thousands of objects, estimating their state vectors and predicting potential intercept and impact points involves four software components that implement distinctly different types of processing requirements.

a. Correlate - Processing of detection data from many sensors of several types, and relating current detections with prior detections, or relating new clusters or objects to prior clusters.

b. Initiate Track - Use of a priori data and logical schema to establish track on objects.

c. Estimate State - Typically, computational-intensive processing of multi-state filters to estimate state of object or clusters of objects.

d. Predict - Forward propagation of object to intercept and earth impact. This requires an for iterative solution for intercept point.

2.2.2.4 Communicate

Communication is a critical complex function that impacts several areas. The categories listed below illustrate the primary areas of SDS communication.

a. Inter-platform data communication - Distribution of extremely large amounts of data between large numbers of space based platforms in a dynamically changing network topology present unique software development requirements.

b. Ground-space communication - This integrates the ground based command and control function with the sensors, weapon platform and the space based battle managers. Such

data communication has unique requirements in that communication must be fault tolerant, highly reliable (in a hostile environment) and trusted/secure.

c. Ground communication - This integrates the various command and control (C^2) centers and ground based weapon installations. Although the security and reliability requirements are less stringent, this type of communication is highly distributed among diverse heterogeneous networks. Of primary importance is interoperability and flexibility.

2.2.2.5 Assess Situation

Because of the requirement to support weapon release, weapon launch, and weapon abort, all aspects of this function are critical to SDS operation. Characterization of software differs between the following two components.

a. Assess Threat - Logically intensive software that must deduce situation from a mixture of a priori data, intelligence data, and surveillance information internal to SDS.

b. Assess SDS - Software that uses available status information available within SDS to support optimal allocation of SDS resources in battle.

2.2.2.6 Assign and Control Weapons

These functions fall into two basic categories:

a. Assign and Control Space Based Interceptor (SBI) Weapons - Because of high absentee ratio, this software component must optimize weapon utilization over a wide range of situations. This component requires rapid response (especially for boost phase intercepts), and an ability to operate in a target-rich or weapon-rich environment but hedge against a shift between these alternatives, and must be able to implement an adaptive preferential strategy within these constraints.

b. Assign and Control Ground Based Interceptor (GBI) Weapons - Because of known interceptor coverage capability and relatively late intercept times, this ground-based component has a less stressing timeline, and must implement preferential target coverage.

2.2.2.7 Guide and Control Weapons

Components of this function differ similarly to the Assign and Control Weapons function.

a. Guide and Control SBI Weapons

b. Guide and Control GBI Weapons

2.2.2.8 Control Platforms

Components of this function are required to provide the housekeeping and control of space-based, and ground-launched (probe) sensor and weapon platforms.

a. Command Environment Control - Command Center Component;

b. Control Onboard Environment - Spaceborne Component;

c. Command Attitude and Position Control - Command Center Component;

d. Control Onboard Attitude and Position - Spaceborne Component;

e. Sense Onboard Status - Spaceborne Component;

f. Assess Status - Command Center Component;

g. Command Reconfiguration - Command Center Component;

h. Reconfigure - Spaceborne Component.

2.2.2.9 Simulate

Simulations are used throughout the Life Cycle of the SDS software. Earliest use supports development of system and software requirements, including system logic and equations. Various simulations are used during software development such as:

a. Development tools - These include timing and performance estimation simulations, threat and environment emulators, etc. Development simulations include non-real-time and real-time software;

b. Hardware-in-the-loop (HWIL) simulators - These are used to troubleshoot interface software and embedded operational software, and are used in the performance of acceptance testing;

c. Demonstration Simulations - Finally, simulations are used to interact with SDS experiments, including those performed with the National Test Bed (NTB).

2.2.2.10 Support Development

Software components are required to support rapid prototyping, or to support unit level, CSC, and CSCI testing during WF software development. Two general components exist, as follows:

a. Support Development Test - Typically this component includes software developed specifically to support testing - drivers, post-processors, etc;

b. Provide Development Environment - This component may include commercial off-the-shelf (COTS), modified, and special software, and supports the coding and de-bugging process.

2.2.2.11 Support Acquisition

During fabrication and delivery of System Element Hardware, support software is required for testing of hardware.

a. Support Factory Test - Typically, this component is a mix of COTS and developed software that is hosted in a wide variety of test equipment, both general-purpose, and peculiar to each tested hardware configuration item;

b. Support Acceptance Test - This component is typically developed software involved in the support of an integrated test of an assembly, platform, or system that demonstrates that it meets requirements for acceptance by the customer.

2.2.2.12 Support Management

A wide variety of COTS and developed software is used in the management of software development. Characterization of the software can be differentiated between two categories, as follows:

a. Maintain and Control Management Information Database - Emphasis in this system is on control and assured backup.

b. Tracking - COTS and developed applications that interface with a variety of users and the database(s).

2.3 CHARACTERISTICS OF SOFTWARE

The list of software characteristics which we first analyzed included sixteen elements. Our initial goal was to reduce that list to a maximum of ten elements. As the result of our effort to date, the proposed list has been pared down to only eight elements. The set is almost orthogonal, with minimum overlap among elements, which is the property we used for scaling down the original number. As an example, the original list had two entries, one for embedded, another for real time. We decided to combine the two entries into one, embedded, because it can be stated that the largest percentage of embedded applications imply real time execution. Another example was to combine available experience with hardware maturity and availability and with technology risk. Lack of experience, and immature hardware are in fact reasons for technology risk. As a result they were all combined in a single characteristic called technology risk.

The final minimum set of software characteristics which affect the quality or the productivity requirements, or the metrics scope are:

a) Criticality;

b) Embedded vs. general purpose;

c) Space/Ground based;

d) Life cycle;

e) Algorithmic Content;

f) Size;

g) Risk; and

h) Intended use.

This section describes the final minimum set of software characteristics which, in turn, affect the requirements of quality attributes. The technique of requiring most quality factors for most software products is very tempting on the surface, as it appears to be a safe policy with no negative effects on the software process or product. However, this is not the case. It is important to maintain the sets of quality factors to a minimum because it:

a) enhances the cost effectiveness of the metrics;

b) focuses on the quality factors which are really critical for that product; and

c) may eliminate or minimize application of conflicting requirements.

It must also be considered that each software product has several characteristics, and the quality factors applicable to a product are the combination of the quality factors applicable to each of the product's characteristics. As an example, assume that the characteristics of a software product are: critical, embedded, and high risk. Then the applicable quality factors are:

a) reliability, survivability, integrity (due to criticality);

b) efficiency and throughput (due to embeddedness);

c) maintainability, usability, portability (due to risk).

The proposed allocation scheme is intended to provide guidelines, not firm requirements, for the selection of quality factors. That selection must be performed by a team of experts composed of users, procurement personnel, and developers, during the earliest development phases, on a product by product basis. The following considerations are intended to be a good starting point for the negotiation process.

A single Computer Software Configuration Item (CSCI) may have Computer Software Components (CSC) which have different characteristics, and therefore may require a different set of quality targets. The methodology then, of assigning quality targets to software must take into account the specific requirements of each CSC. If this is not done, then overdesigned or underdesigned software may result. As noted above, the guidelines are intended to be just that. Specific products may require deviations from the guidelines. To date, we have

broken down major SDS functions to elements which, although more typically at the CSCI level than the CSC level, can be characterized adequately to apply quality attributes. The following set of software characteristics follow generally but have been modified somewhat from the set furnished in the outline for this report.

2.3.1 Criticality

Criticality is a most important software characteristic, which applies primarily to operational software, but also to elements of the support software. Key characteristics to be considered in determining level of criticality are whether the function and timing are critical (such as in real-time signal processing of sensor detections), where the function is critical but the timing less critical (software that drives command center status update displays) or non-critical (management report generation applications, for instance). The relevant quality factors are: reliability, survivability, and integrity. Metrics elements supporting those quality factors must encompass a broad range of development and verification activities including error rate detection and discrepancy report and discrepancy resolution records. We must realize, however, that the technology for predicting operational failure rates, based on development effort, error rate, or other techniques is not available yet, as previously discussed. The importance of establishing research programs for satisfying these technology needs, can not be underestimated, in our opinion.

2.3.2 Embedded vs. general purpose (real/non real time)

Embedded software executes in real time, and it interfaces with external equipment and possibly operational users. The primary quality factors for embedded software are throughput and efficiency. It is important to have computational and transmission duty cycles not higher than 60% because:

a) the computational requirements may grow beyond the original estimates; and

b) it has been consistently demonstrated that the failure rate of real time systems tends to increase as the computational and transmission duty cycle approaches 100%.

Highly efficient software may be desirable, for the above considerations. It must be realized, however, that high throughput can not be achieved at the cost of other quality factors.

like maintenance or reusability, if they are also important for that specific product. In those cases, then more efficient hardware must be utilized and developed, if not available. The appropriate compromises must be resolved in a case to case basis.

2.3.3 Space/Ground-Based

The primary quality factors of space based software are:

a) reliability which is required because of the inherent difficulty of maintaining software in space;

b) survivability which is required because of the need of operating in a war environment; and

c) integrity which is required because of the need to be protected relative to unauthorized access.

The issue of integrity is very complex, and the suitability of metrics for dealing with that requirement is uncertain, as it was previously discussed.

2.3.4 Life Cycle

SDS software products have typically a long life cycle. The primary quality factors of products with a long life cycle are:

a) Maintainability. The evolving requirements of the software, the needs of upgrading and fixing bugs require high level of maintainability;

b) Usability, primarily in the area of ease of de-bugging and modification. Several generations of users will interface with the software and easy of use and learning may be very cost and operational effective; and

c) Portability. It may be important for the software to be portable to other environments and of being capable of interfacing with a variety of other software products.

The relevance of the three identified requirements is expected to be increasingly higher as a function of the life cycle length. SDS software will be developed and used throughout the system life cycle, including development, acquisition, deployment, and operational phases.

The importance of usability, maintainability, and portability depends on the extent of the life cycle. The intended use of the software may be an important modifier of that relevance.

2.3.5 Algorithmic Content

Software may be viewed as having higher or lower levels of

a) logic computations; and

b) mathematical equation computation.

Quality attributes include Survivability (ability to deal with decision conflicts and data inconsistency, for example), and Throughput (ability to process large numbers of matrix calculations in real time).

2.3.6 Size

Cost increases proportionally with size, all other factors being equal. The primary quality factors of large, high cost, programs are Maintainability, Integrity, and Reusability. Large programs require large maintenance cost, which can be significantly reduced if the programs are easy to maintain, verify, expand, etc. The assumption made here is that large programs also have long life cycles, which reinforce the need for high maintainability. Development of very large programs mean relatively large numbers of people involved throughout the life cycle of the software, placing a high level of importance on integrity across the life cycle, and between many CSCs. Large programs also often include capabilities which can be reused, and they can themselves include software previously designed. However, reuse may be not very significant in the case of embedded software which often has environmental constraints and hardware dependencies which make it difficult to reuse.

2.3.7 Risk

A software program can be a high risk item because the available experience on the application is low, the hardware within the hosting environment is not developed and mature, or the required performance far exceeds previous experiences. High risk software is a category for which RP development approach may be very effective, as it was previously described. Independently from

the development approach, however, high risk software most likely requires an evolutionary development which may require several requirements and design specification changes. For this reason primary quality factors for high risk software are:

a) Maintainability, so that design and implementation changes can be easily implemented and verified. Product requirements and characteristics will most likely evolve during the development process which will, in turn, require modifications to be implemented and effects to be analyzed;

b) Usability, so that a large number of users/developers can easily exercise it and evaluate it. The software will likely be exercised and analyzed by personnel with different backgrounds, some with little or no experience in computer science;

c) Portability, so that can be interfaced with different products for evaluation and hosted in different environments. Also, as noted in b) above, it may be required to be exercised by different groups of people in different environments.

Quality factors such as integrity, reliability, and survivability, on the other hand, are required in the case of critical or space based products which may, or may not, have high development risk. As such, they are not in and of themselves prime factors for high risk product development.

2.3.8 Intended Use

SDS software can be divided in the following categories:

a) operational software, which includes all the software which is required for mission success;

b) simulations of operational scenarios and emulations of technology;

c) development and verification environments and tools; and

d) management tools, like the metrics.

Major discriminants for quality include whether the software is to be used in peacetime, during battle, or both; whether it is related to the use of weapons or not (since weapon control software has a higher level of criticality); and whether the software is operational or non-

operational. Each category has distinct requirements relative to quality metrics. Operational software typically has a high degree of criticality (see next paragraph). Development software may include off the shelves products, like compilers, environments, general purpose tools etc. Most simulations are designed for validating concepts and estimating effectiveness of configurations. Management tools include configuration management and control, data bases, and metrics. Primary quality factors for support (i.e., non-operational) software are maintainability and usability. Such software needs to be easy to use, as many different users with a wide range of backgrounds and interests will utilize the software. It must also be easy to maintain, as during its useful life, support software will often be enhanced to include capabilities in addition to those originally implemented. Note that the application of development metrics for predicting the user quality ratings during the development process of a product, is not necessary for off-the-shelf software. That software can be directly exercised and evaluated relative to the user requirements, which eliminates the need for predicting quality factors. Development software is often less critical than operational software. Some development tools may be very critical, however, like in the case of Ada compilers, which must be validated to very strict standards.

The need of defining intended use as a software characteristic is not clearly established because intended use is not completely orthogonal relative to the other characteristics previously described, such as Criticality. Our proposed approach is to include it, at least initially. It can be later removed if proven unnecessary.

2.4 PRELIMINARY REQUIREMENTS FOR SOFTWARE QUALITY DOMAINS

The impact that each characteristic has on the metrics is discussed below. Each combination of characteristics defines the needs for a software domain with unique quality requirements, which are represented by a unique combination of the following quality factors: Reliability, Efficiency, Throughput Integrity, Survivability, Usability, Maintainability, Portability, and Reuse. Based on the application of these factors to the software characteristics, a relative level of importance for software metrics is defined for each quality attribute.

2.4.1 Detect Plumes

- Level of Criticality; time and function critical;

- Embedded real-time;
- Space Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; equation-computation intensive;
- Size; small;
- Technology Risk; moderate;
- Intended Use; Peacetime and Battle, non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	High
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.2 Detect Cold Bodies

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based or Probe-based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic-intensive, equation-computation intensive;
- Size; large;
- Technology Risk; moderate;
- Intended Use; Battle, non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	High
Throughput	High
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.3 RF Detect

- Level of Criticality; time and function critical;
- Embedded real-time;
- Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic-intensive, equation-computation intensive;
- Size; large;
- Technology Risk; moderate;
- Intended Use; Peacetime and Battle, Weapon-related and non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	High
Usability	Low
Maintainability	Moderate
Portability	Moderate
Reuse	Low

2.4.4 Resolve Objects

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic-intensive, equation-computation intensive;
- Size; moderate;
- Technology Risk; moderate;
- Intended Use; Battle, non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.5 Discriminate

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based and/or Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic-intensive, equation-computation intensive;
- Size; large;

- Technology Risk; high;
- Intended Use; Battle, non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.6 Assess Kills

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based and/or Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic-intensive;
- Size; moderate;
- Technology Risk; moderate;
- Intended Use; Battle, Weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Moderate
Maintainability	Moderate
Portability	Moderate
Reuse	Low

2.4.7 Correlate

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based and/or Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic and equation-computation intensive;
- Size; small;
- Technology Risk; moderate;
- Intended Use; Battle, Weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	High
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.8 Initiate Track

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based and/or Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic-intensive;
- Size; small;

- Technology Risk; moderate;
- Intended Use; Battle, Weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	High
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.9 Estimate State

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based and/or Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; equation-computation intensive;
- Size; small;
- Technology Risk; moderate;
- Intended Use; Battle, Weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	High
Usability	Low
Maintainability	Low
Portability	Moderate
Reuse	Low

2.4.10 Predict Intercept and Impact Points

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based and/or Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; equation-computation intensive;
- Size; small;
- Technology Risk; low;
- Intended Use; Battle, Weapon-related vs. non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.11 Interplatform Data Communication

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic-intensive (network);
- Size; moderate (highly distributed);

- Technology Risk; very high;
- Intended Use; Peacetime and Battle, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	High
Throughput	High
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Moderate

2.4.12 Ground - Space Communication

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based and Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic-intensive (computation, encryption);
- Size; small;
- Technology Risk; low;
- Intended Use; Peacetime and Battle, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Low

Maintainability	Moderate
Portability	Moderate
Reuse	Low

2.4.13 Ground Communication

- Level of Criticality; time and function critical;
- Embedded real-time;
- Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic-intensive;
- Size; moderate;
- Technology Risk; low;
- Intended Use; Peacetime and Battle, operational.

Quality Attributes:	Relative Ranking:
Reliability	Moderate
Survivability	Moderate
Integrity	Moderate
Efficiency	Low
Throughput	Moderate
Usability	Moderate
Maintainability	High
Portability	Moderate
Reuse	High

2.4.14 Assess Threat

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based and/or Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic-intensive;

- Size; small;
- Technology Risk; moderate;
- Intended Use; Peacetime and Battle, Weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Low
Throughput	Low
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.15 Assess SDS

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based and/or Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic-intensive;
- Size; small;
- Technology Risk; low;
- Intended Use; Peacetime and Battle, Weapon-related vs. non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Low
Throughput	Low

Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.16 Assign and Control SBI Weapons

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; logic-intensive, equation-computation intensive;
- Size; moderate;
- Technology Risk; moderate;
- Intended Use; Battle, Weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	High
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.17 Assign and Control GBI Weapons

- Level of Criticality; time and function critical;
- Embedded real-time;
- Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;

- Algorithmic Content; equation-computation intensive;
- Size; small;
- Technology Risk; moderate;
- Intended Use; Battle, Weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Low
Maintainability	High
Portability	Low
Reuse	Low

2.4.18 Guide and Control SBI Weapons

- Level of Criticality; time and function critical;
- Embedded real-time;
- Space Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; equation-computation intensive;
- Size; small;
- Technology Risk; moderate;
- Intended Use; Battle, Weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	High
Throughput	High

Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.19 Guide and Control GBI Weapons

- Level of Criticality; time and function critical;
- Embedded real-time;
- Interceptor Based and Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Algorithmic Content; equation-computation intensive;
- Size; small;
- Technology Risk; moderate;
- Intended Use; Battle, Weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Low
Maintainability	High
Portability	Low
Reuse	Low

2.4.20 Command Environment Control

- Level of Criticality; function critical;
- Embedded real-time;
- Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;

- Size small;
- Technology Risk; low
- Intended Use; Peacetime and Battle, Weapon-related and non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	Moderate
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Moderate
Maintainability	High
Portability	Moderate
Reuse	Low

2.4.21 Control Onboard Environment

- Level of Criticality; function critical,
- Embedded real-time;
- Space Based ;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Size; small;
- Technology Risk; low;
- Intended Use; Peacetime and Battle, Weapon-related and non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	High
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.22 Command Attitude and Position Control

- Level of Criticality; function critical;
- Embedded real-time;
- Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Size; small;
- Technology Risk; low;
- Intended Use; Peacetime and Battle, Weapon-related and non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	Moderate
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Moderate
Maintainability	High
Portability	Moderate
Reuse	Low

2.4.23 Control Onboard Attitude and Position

- Level of Criticality; function critical;
- Embedded real-time;
- Space Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Size; small;
- Technology Risk; low;

- Intended Use; Peacetime and Battle, Weapon-related and non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	High
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.24 Sense Onboard Status

- Level of Criticality; function critical;
- Embedded real-time;
- Space Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Size; small;
- Technology Risk; low;
- Intended Use; Peacetime and Battle, Weapon-related and non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.25 Assess Status

- Level of Criticality; function critical;
- Embedded real-time;
- Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Size; small;
- Technology Risk; low;
- Intended Use; Peacetime and Battle, Weapon-related and non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	Moderate
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Moderate
Maintainability	High
Portability	Moderate
Reuse	Low

2.4.26 Command Reconfiguration

- Level of Criticality; function critical;
- Embedded real-time;
- Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Size; small;
- Technology Risk; low;
- Intended Use; Peacetime and Battle, Weapon-related and non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	Moderate
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Moderate
Maintainability	High
Portability	Moderate
Reuse	Low

2.4.27 Reconfigure

- Level of Criticality; function critical;
- Embedded real-time;
- Space Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Size; small;
- Technology Risk; low;
- Intended Use; Peacetime and Battle, Weapon-related and non-weapon-related, operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	High
Usability	Low
Maintainability	Low
Portability	Low
Reuse	Low

2.4.28 Development Tools

- Level of Criticality; typically, time or function critical;

- Real-time and Non-real-time General Purpose;
- Ground Based;
- Length of Life Cycle; through development;
- Algorithmic Content; logic-intensive, equation-computation intensive;
- Size; large;
- Technology Risk; low;
- Intended Use; non-operational.

Quality Attributes:	Relative Ranking:
Reliability	Moderate
Survivability	Moderate
Integrity	Moderate
Efficiency	Moderate
Throughput	Low
Usability	Moderate
Maintainability	Moderate
Portability	Moderate
Reuse	Moderate

2.4.29 Hardware-in-the-loop (HWIL) Simulators

- Level of Criticality; time and function critical;
- Embedded real-time;
- Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation (includes operational trouble-shooting);
- Algorithmic Content; logic-intensive, equation-computation intensive;
- Size; large;
- Technology Risk; moderate;
- Intended Use; Peacetime, non-weapon-related, non-operational.

Quality Attributes:	Relative Ranking:
Reliability	Moderate
Survivability	High
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	Moderate
Maintainability	Moderate
Portability	Low
Reuse	Low

2.4.30 Demonstration Simulations

- Level of Criticality; time and function critical;
- Embedded and other real-time and Near-real-time;
- Ground Based;
- Length of Life Cycle; through development;
- Algorithmic Content; logic-intensive, equation-computation intensive;
- Size; large;
- Technology Risk; moderate;
- Intended Use; non-operational.

Quality Attributes:	Relative Ranking:
Reliability	Moderate
Survivability	High
Integrity	Moderate
Efficiency	Moderate
Throughput	Moderate
Usability	Moderate
Maintainability	Moderate
Portability	Moderate
Reuse	Low

2.4.31 Support Development Test

- Level of Criticality; function critical;

- Real-time and Non-real-time General Purpose;
- Ground Based;
- Length of Life Cycle; through development;
- Size; large;
- Technology Risk; moderate;
- Intended Use; non-operational.

Quality Attributes:	Relative Ranking:
Reliability	Moderate
Survivability	Moderate
Integrity	Moderate
Efficiency	Moderate
Throughput	Moderate
Usability	High
Maintainability	High
Portability	High
Reuse	High

2.4.32 Provide Development Environment

- Level of Criticality; function critical;
- Embedded real-time and Non-real-time General Purpose;
- Ground Based;
- Length of Life Cycle; through development, acquisition, deployment, operation;
- Size; moderate;
- Technology Risk; low;
- Intended Use; non-operational.

Quality Attributes:	Relative Ranking:
Reliability	High
Survivability	Moderate
Integrity	High

Efficiency	Moderate
Throughput	Moderate
Usability	High
Maintainability	Moderate
Portability	Low
Reuse	Low

2.4.33 Support Factory Test

- Level of Criticality; function critical;
- Non-real-time General Purpose;
- Size; large;
- Technology Risk; low;
- Intended Use; non-operational.

Quality Attributes:	Relative Ranking:
Reliability	Moderate
Survivability	Moderate
Integrity	Moderate
Efficiency	Low
Throughput	Low
Usability	High
Maintainability	High
Portability	High
Reuse	High

2.4.34 Support Acceptance Test

- Level of Criticality; function critical;
- Real-time and Non-real-time General Purpose;
- Length of Life Cycle; through acquisition;
- Size; large;
- Technology Risk; low;
- Intended Use; non-operational.

Quality Attributes:	Relative Ranking:
Reliability	Moderate
Survivability	Moderate
Integrity	Moderate
Efficiency	Moderate
Throughput	Moderate
Usability	Low
Maintainability	Moderate
Portability	Low
Reuse	Low

2.4.35 Maintain and Control Management Information Database

- Level of Criticality; function critical;
- Non-real-time General Purpose;
- Length of Life Cycle; through development, acquisition;
- Technology Risk; low;
- Intended Use; Peacetime, non-weapon-related, non-operational.

Quality Attributes:	Relative Ranking:
Reliability	Moderate
Survivability	Moderate
Integrity	High
Efficiency	Moderate
Throughput	Moderate
Usability	High
Maintainability	High
Portability	High
Reuse	High

2.4.36 Management Information Tracking

- Non-real-time General Purpose;
- Length of Life Cycle; through development, acquisition;
- Technology Risk; low;
- Intended Use; Peacetime, non-weapon-related, non-operational.

Quality Attributes:	Relative Ranking:
Reliability	Moderate
Survivability	Low
Integrity	High
Efficiency	Moderate
Throughput	Low
Usability	High
Maintainability	High
Portability	High
Reuse	High

Figure 2-1 summarizes the quality factor ranking for all SDS software functions which have been identified.

2.5 PROCESS TYPING

In this section of the document the characteristics of the software products which affect the development process are discussed. These characteristics include:

a) Criticality and predicted size and development cost; and

b) Technology risks.

The former characteristics determine the most appropriate and cost effective development, auditing, and reviewing level of effort. The latter determine the most appropriate development process model.

The development of software products which are mission critical must fully comply with the highest level of formality which is specified for the model of development process best suited for that product. The level of documentation must fully support all the requirements of the scheduled auditing and review activities. Software metrics must also be conducted consistently with the highest standard specified and must be integrated with, and must complement the other auditing activities. It is essential to monitor the reliability, survivability and integrity metrics, by accurately and consistently collecting all discrepancy reports, and their resolution.

The development of large software programs is a very costly process, independent from the criticality of such programs. They need to be developed to the highest standard of formality so that progress and cost can be matched accurately and overruns can be avoided or at

FUNCTION \ QUALITY ATTRIBUTE RELATIVE RANKING	RELIABILITY	SURVIVABILITY	INTEGRITY	THROUGHPUT	USABILITY	MAINTAINABILITY	PORTABILITY	REUSE	EFFICIENCY
2.4.1 DETECT PLUMES	H	H	H	H	L	L	L	L	M
2.4.2 DETECT COLD BODIES	H	H	H	H	L	L	L	L	H
2.4.3 RF DETECT	H	H	H	H	L	M	L	L	M
2.4.4 RESOLVE OBJECTS	H	H	H	M	L	L	M	L	M
2.4.5 DISCRIMINATE	H	H	H	M	L	L	L	L	M
2.4.6 ASSESS KILLS	H	H	H	M	M	M	L	L	M
2.4.7 CORRELATE	H	H	H	H	L	L	M	L	M
2.4.8 INITIATE TRACK	H	H	H	H	L	L	L	L	M
2.4.9 ESTIMATE STATE	H	H	H	H	L	L	L	L	M
2.4.10 PREDICT INTERCEPT & IMPACT POINTS	H	H	H	M	L	L	M	L	M
2.4.11 INTERPLATFORM DATA COMMUNICATION	H	H	H	H	L	L	L	M	H
2.4.12 GROUND-SPACE COMMUNICATION	H	H	H	M	L	M	L	L	M
2.4.13 GROUND COMMUNICATION	M	M	M	L	M	H	L	H	L
2.4.14 ASSESS THREAT	H	H	H	L	L	L	M	L	L
2.4.15 ASSESS SDS	H	H	H	L	L	L	L	L	L
2.4.16 ASSIGN & CONTROL SBI WEAPONS	H	H	H	H	L	L	L	L	M
2.4.17 ASSIGN & CONTROL GBI WEAPONS	H	H	H	M	L	H	L	L	M
2.4.18 GUIDE & CONTROL SBI WEAPONS	H	H	H	H	L	L	L	L	H
2.4.19 GUIDE & CONTROL GBI WEAPONS	H	H	H	M	L	H	L	L	M
2.4.20 COMMAND ENVIRONMENT CONTROL	H	M	H	M	M	H	L	L	M
2.4.21 CONTROL ONBOARD ENVIRONMENT	H	H	H	H	L	L	M	L	M
2.4.22 COMMAND ATTITUDE & POSTION CONTROL	H	M	H	M	M	H	L	L	M
2.4.23 CONTROL ONBOARD ATTITUDE & POSTION	H	H	H	H	L	L	M	L	M
2.4.24 SENSE ONBOARD STATUS	H	H	H	M	L	L	L	L	M
2.4.25 ASSESS STATUS	H	M	H	M	M	H	L	L	M
2.4.26 COMMAND RECONFIGURATION	H	M	H	M	M	H	M	L	M
2.4.27 RECONFIGURE	H	H	H	H	L	L	L	L	M
2.4.28 DEVELOPMENT TOOLS	M	M	M	L	M	M	M	M	M
2.4.29 HWIL SIMULATOR	M	H	H	M	M	M	L	L	M
2.4.30 DEMONSTRATION SIMULATIONS	M	H	M	M	M	M	M	L	M
2.4.31 SUPPORT DEVELOPMENT TEST	M	M	M	M	H	H	H	H	M
2.4.32 PROVIDE DEVELOPMENT ENVIRONMENT	H	M	H	M	H	M	L	L	M
2.4.33 SUPPORT FACTORY TEST	M	M	M	L	H	H	H	H	L
2.4.34 SUPPORT ACCEPTANCE TEST	M	M	M	M	L	M	L	L	M
2.4.35 MAINTAIN & CONTROL MGMT INFO DATABASE	M	M	H	M	H	H	H	H	M
2.4.36 MANAGEMENT INFORMATION TRACKING	M	L	H	L	H	H	H	H	M

0389/002-001

Figure 2-1 Preliminary Requirements for Software Quality

least minimized. The focus of the activities must be centered on these objectives. Productivity metrics provide effective means for matching progress and incurred expenses. They are discussed in Section 2.6.

Criticality, size and cost can then be combined to define three levels of software: Level 1, 2, and 3. Each level differs from the others by a combination of criticality, size and cost, (Level 1 is the most critical, expensive, etc.; and Level 3 the least critical, expensive, etc.). For each level, different auditing and review activities, including metrics, are specified (See Figure 2-2, Metrics Application and Software Level).

ACTIVITIES/LEVEL	LEVEL 1	LEVEL 2	LEVEL 3
REQUIREMENTS	A	A	C
PRELIM. DES.	A	A	C
DETAIL DES.	A	B	C
CODING & CSU TEST	A	B	
CSC INTEGR. TEST	A	B	
CSCI INTEGR. TEST	A	B	

Figure 2-2 Metrics Application and Software Level

Activity A implies that metrics must be enforced to demonstrate compliance with preset target values of all quality factors. Level 1 software requires such activities at the end of all major development phases. Activity B requires a lesser degree of auditing and reviews, which is limited to the most critical quality factors only. Level 2 software requires a combination of A and B activities, at different phases of the development process. Finally, level 3 software, the least critical, expensive or complex, and the smallest in size, may require very limited or no metrics activity at all. It is important to establish different levels of software because auditing and review activities, including metrics, are costly and it is not cost effective to equally and indiscriminately apply them to all software.

Classifications of this kind have been successfully used for supporting the development process of the software embedded in digital flight control systems with different

levels of criticality and complexity. In that case, the criticality of the software was determined, very early in the development process, by mutual agreement of manufacturers and responsible government agencies. The same process is suggested for SDS software.

In the previous sections of this document it was discussed why the RP development process is particularly effective in the case of large complex user interfaces, and in the case of high technology risk areas. Clearly an early decision must be made, by mutual consent, as to the most cost effective development approach for each particular program. In the case that RP is selected, a clear definition of development and auditing activities must be outlined. As it was previously discussed, the methodology supporting the RP process is not well developed. It is strongly suggested to initiate a program which addresses this specific issue.

2.6 PRODUCTIVITY MEASUREMENT REQUIREMENTS

The questions which productivity metrics attempt to answer are:

a) Is the program on schedule?

b) Is the program on budget?

c) What changes to the development process can be made to improve productivity?

Uniform answers are difficult to obtain because of many different factors, including different development processes, resource availability, attributes of the products, and, most importantly, large differences in engineer/programmer productivity due to personal motivation, job satisfaction, capability, and other factors difficult to quantify. A source of the confusion which can and must be eliminated is that which results from the lack of a standard taxonomy. As an example, two simple measures of productivity include developed lines of code (SLOC), and SLOC per man-hour. However, SLOC is not uniformly used, and different measures result from taking into account comments and data statements in one case, but not another, for example.

The following assertions are widely accepted and have empirical basis:

a) Most errors are introduced in the earliest phases of the development process;

b) The cost of correcting errors increases as a function of the age of the error; that is, the elapsed time from error insertion to error detection and correction.

A study done by the Software Engineering Laboratory of NASA examining the matter ("Metric Analysis and Data Validation Across FORTRAN Projects", IEEE Transactions on Software Engineering, Vol. SE-9, No. 6, Nov. 1983) shows that neither Software Science, nor Cyclomatic Number, nor SLOC correlate well to effort spent or errors incurred. Correlations, however, improve when the variability of the products and the development environment decreases. This basically proves that productivity metrics, like quality metrics, must be tailored to the specific project.

Error rate trend appears to be a promising technique for estimating the maturity of the software during the development process. However, there is not at this time a clear indication that error rate trend during development is a good predictor of operational reliability. A study is needed to determine if that correlation can be improved by appropriately tailoring the software domain. During this study, a distinction must be made between errors which may prevent the successful completion of the mission and errors which do not affect the mission. It is essential to develop a methodology for addressing the former critical class of errors.

This section of the document describes which methods are currently available for:

a) estimating the system size

b) estimating development cost

c) estimating completion date.

A qualitative methodology for identifying causes of deviations from predicted patterns is also described. The contents of this section of the document is closely related to the work done in the Software Engineering Lab (SEL) of NASA/Goddard Space Flight Center (GSFC).

2.6.1 Prediction of Program Size

It is very important to accurately predict software size because most other required characteristics depend on predicted size. The best tool for predicting size is past experience with projects of

similar requirements and constraints. The estimation of the size evolves during the development process, and it becomes more detailed and more accurate as more information is available.

In a technical interchange meeting with the SDS System Engineer (General Electric) two recommended methods were presented for size estimation - Analogy Analysis and Function Point Analysis.

Analogy analysis relies upon collecting empirical data from similar programs in an attempt to derive estimates based similarities found in previous efforts. The advantage of this approach is that it may be applied at a very early phase of the system life cycle. However, the large number of variables involved in a specific development effort make it very difficult to determine projected parallels between a planned development and a completed project. Unless the correlation between the two projects' characteristics is very high, the confidence interval of the size estimate is likely to be very limited. In addition, the measurands must have been generated using a standard set of assumptions. For example, in order to draw valid analogies concerning the estimated size in source lines of code for functions, the implementation language should be the same. In the Ada language, the Government and industry have not yet agreed on standards on what constitutes a single line of source code. Unfortunately, standards used for generating measurement data across empirical databases of prior projects may not agree (and in some instances may not be known). A less formal invocation of the Analogy method would be to make use of application experts with long term prior experience with similar systems. The specific model for Analogy Analysis currently recommended by the SE is SSM.

The other technique suggested by the SE is the Function Point method. With this method, estimates are derived via a parametric based expert system algorithm of SLOC "factors" for like functions. This, however, requires a significant level of detail for the requirements in order to obtain estimates with high degrees of confidence. It is likely to be more useful in mid to later stages of the system life cycle (i.e. after SRR). The specific Function Point model recommended by the SE is ASSET-R. The SE currently recommends the use of both models depending upon the stage of the system life cycle in which they applied.

2.6.2 Estimation of Program Cost

Size is clearly a major factor in determining cost. In fact, most cost models are expressed in terms of SLOC. A typical equation relating cost to program size is:

$$MM = A * (KDSI)^B$$

where:
MM is the amount of the effort in Man-months

KDSI is the number of thousands of delivered source code instructions

A, B are constants for a given program.

A and B vary as a function of many characteristics of the product and the process. The COnstructive COst MOdel (COCOMO) developed by B. Boehm (Software Engineering Economics) defines three categories of product/process combinations, called modes:

a) Organic, the least demanding of the three because it implies personnel experienced with the application and with the environment, and relatively few constraints (an example would be production of development tools)

b) Embedded, the most demanding. The product and the process are tightly constrained by hardware, existing software, and operational procedures. Examples are detection, tracking, and communication.

c) Semidetached, which is an intermediate state between organic and embedded. Examples are simulations.

The values of A and B are the lowest in the case of organic mode, and highest in the case of embedded mode.

MODE	A	B
Organic	2.4	1.05
Semidetached	3.0	1.12
Embedded	3.6	1.20

Several other models exist which use the same relationship between man-months and delivered source instructions. These models differ from each other by the numerical values used for the constants A and B. The methodology which we propose is based on the tailoring of

that relationship to the same software domains which have been identified for the software quality attributes.

A recent study completed by TASC for ESD shows that the cost per source line of code can vary approximately by a factor of six (6), as a function of the application. The lowest cost range of $100 - $150 occurs in the case of non-real-time support utilities; the highest cost range of $700 - $1000 occurs in the case of mission critical, real-time, attack warning software.

The percentage of software reuse which is included is clearly an important parameter for estimating overall development cost. Software re-use is dependent on the availability of the software, the quality of the documentation and support, and the willingness of the developers to reuse it. A cost model must be developed similar to the cost model for new software, but which also includes other parameters that take into account the quality of the documentation, quality of the code as measured by appropriate quality attributes, and support available (the support can be rated as excellent, good, minimum, or none), and the amount of required changes. Three levels of changes may be appropriate:

a) significant changes required (more than 25% of the product needs to be modified);

b) minor changes (less than 25%),

c) no changes are required.

The parameters discussed may be used as modifiers of the cost equation for new software.

The SDS SE has recommended two costing models to be incorporated into the cost estimation activities conducted by the SDS Software Center.

The first, COCOMO, was selected because of its wide acceptance and use in current software projects. However, the SE did note that the accuracy of this model is limited, and that the parameters are still open to discussion/debate.

The second, SoftCost-Ada, is a newer Ada oriented model being developed by RCI. This model is based on a modified Rayleigh-Norden-Putnam formulation using power law

fits of data to determine confidence intervals of estimates for effort, duration, and staffing predictions. Initial applications of this model have been very encouraging.

2.6.3 Estimation of Time to Complete

The initial estimate of time to complete can be based on the overall effort required in man-months and the level of allocated manpower. Past experiences must be used for:

a) Sizing the efforts (staff-months) required to complete each major development activity, such as requirement specifications, preliminary design, etc., and

b) Allocating the proper level of staffing to each development phase.

The overall time to complete can therefore be easily determined, provided that process bottlenecks do not materialize which prevent the achievement of the development milestones in the predetermined sequence.

Past experience must be utilized for building models of the development process. Examples would be life cycle models of the percentage of effort, and schedule, for each development phase. The distribution of effort among the several development phases follow predictable paths which must be defined for each software domain. Models must be developed for "successful" programs. Then deviations from successful models may provide early warnings of pending problems. As an example, a successful model for a certain type of software may imply that the requirements and design phases consume a certain amount of the resources, and of the overall schedule. Then, if code is being developed prior to those resources being consumed, that may be an indication of inadequate design. Some nominal development process patterns have been developed from the SEL database of NASA/GSFC. Examples of nominal range of behavior are shown in Figures 2-3, 2-4, and 2-5 which are assumed to reflect the pattern of "successful" programs. Figure 2-3 shows the nominal range for code production. Code which is produced too early in the development process may be a symptom of inadequate design and/or testing. The converse may be a sign of a development process which is behind schedule. Figure 2-4 shows the nominal range for computer usage. Excessive use in the early phases may be a sign of insufficient design. Low use may be a sign of slow production rate. Figure 2-5 shows the nominal pattern of software changes per SLOCs. Excessive changes may be a sign of poor design, too few changes

may be a sign of insufficient testing. Nominal patterns must be tailored to specific software domains and they may be effectively used to spot suspicious behavior patterns, which may require further analysis. These type of nominal patterns have only qualitative value. Quantitative models must be developed for each SDS software domain.

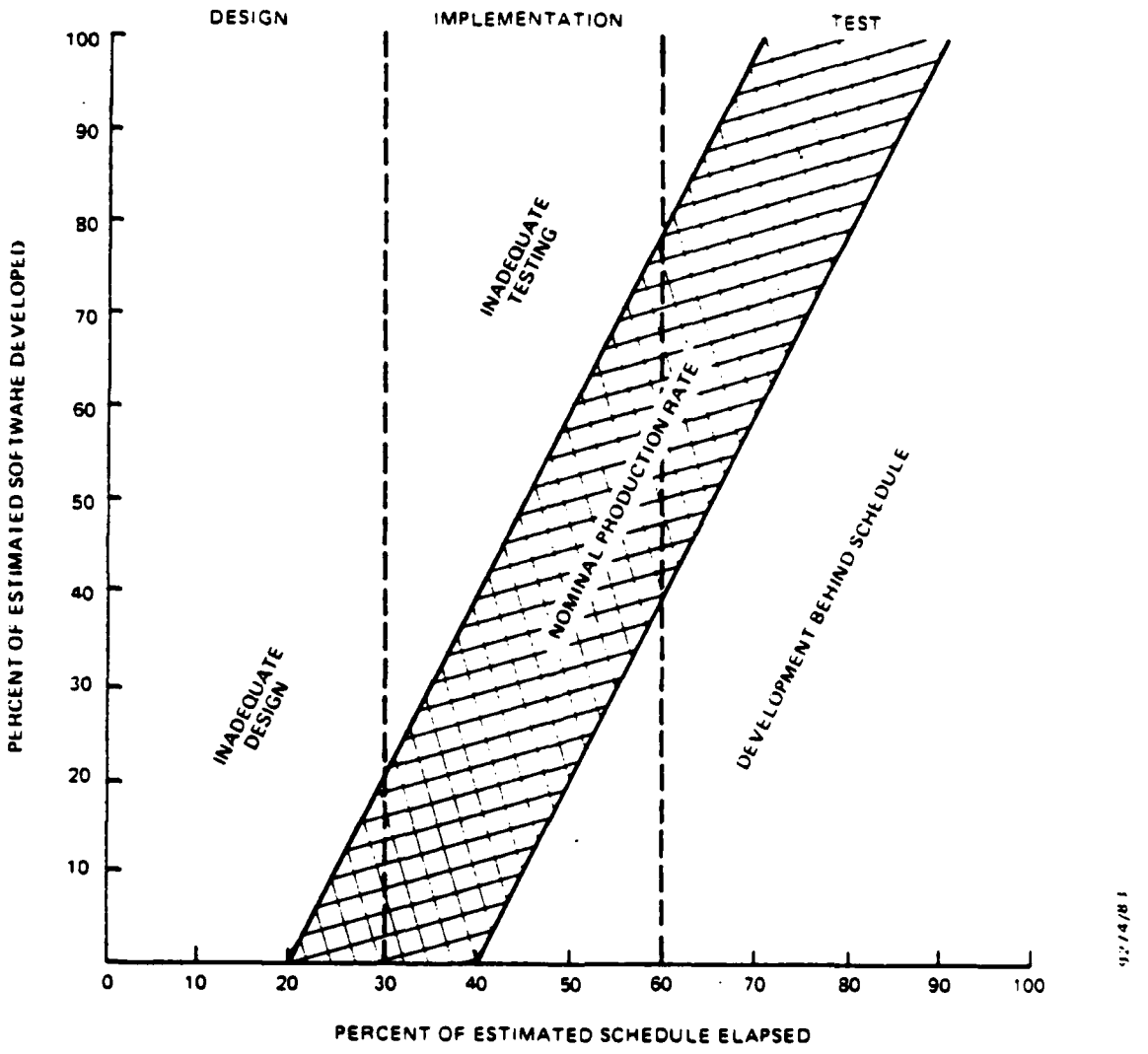


Figure 2-3 Nominal Software Production Pattern

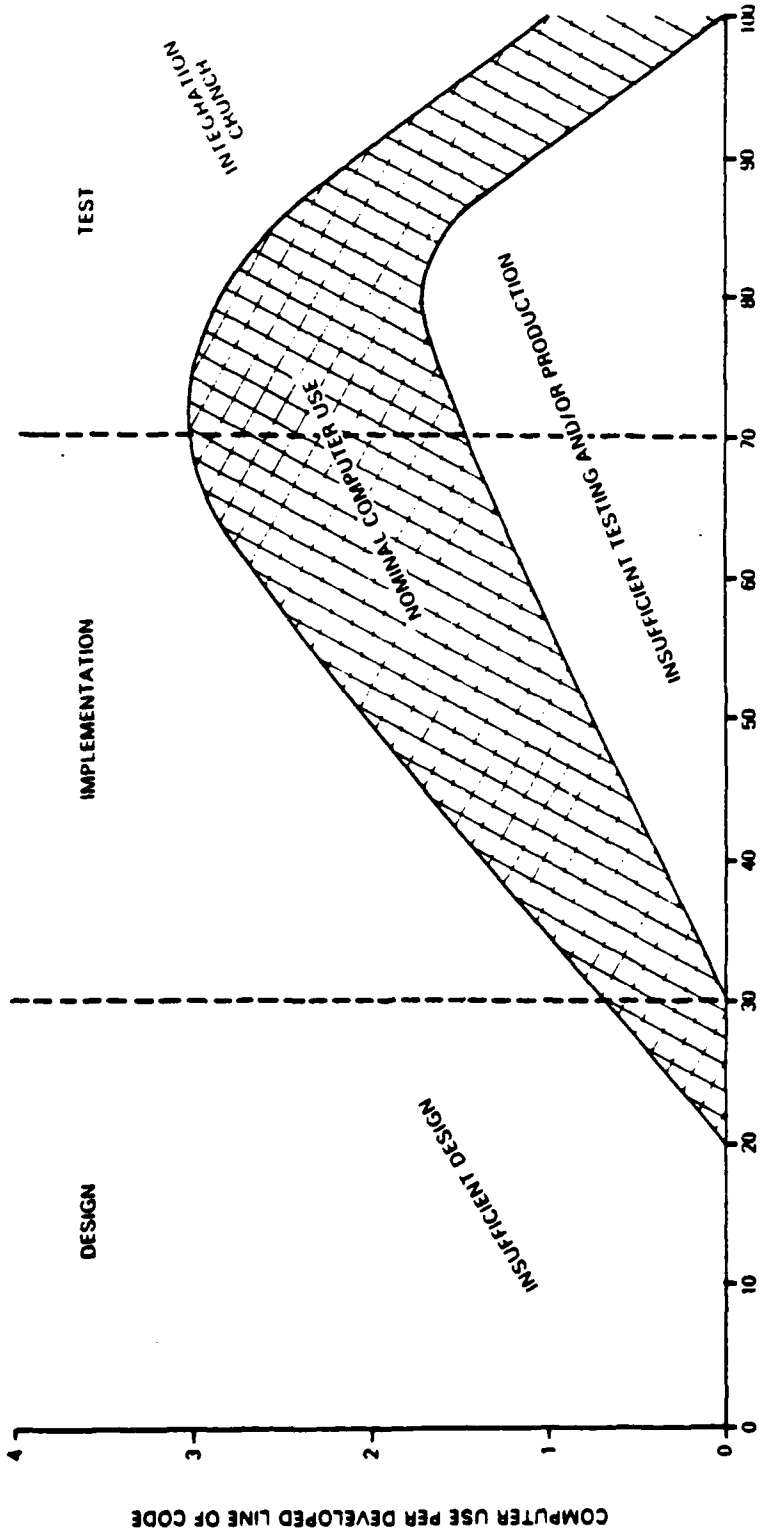


Figure 2-4. Nominal Computer Utilization Pattern

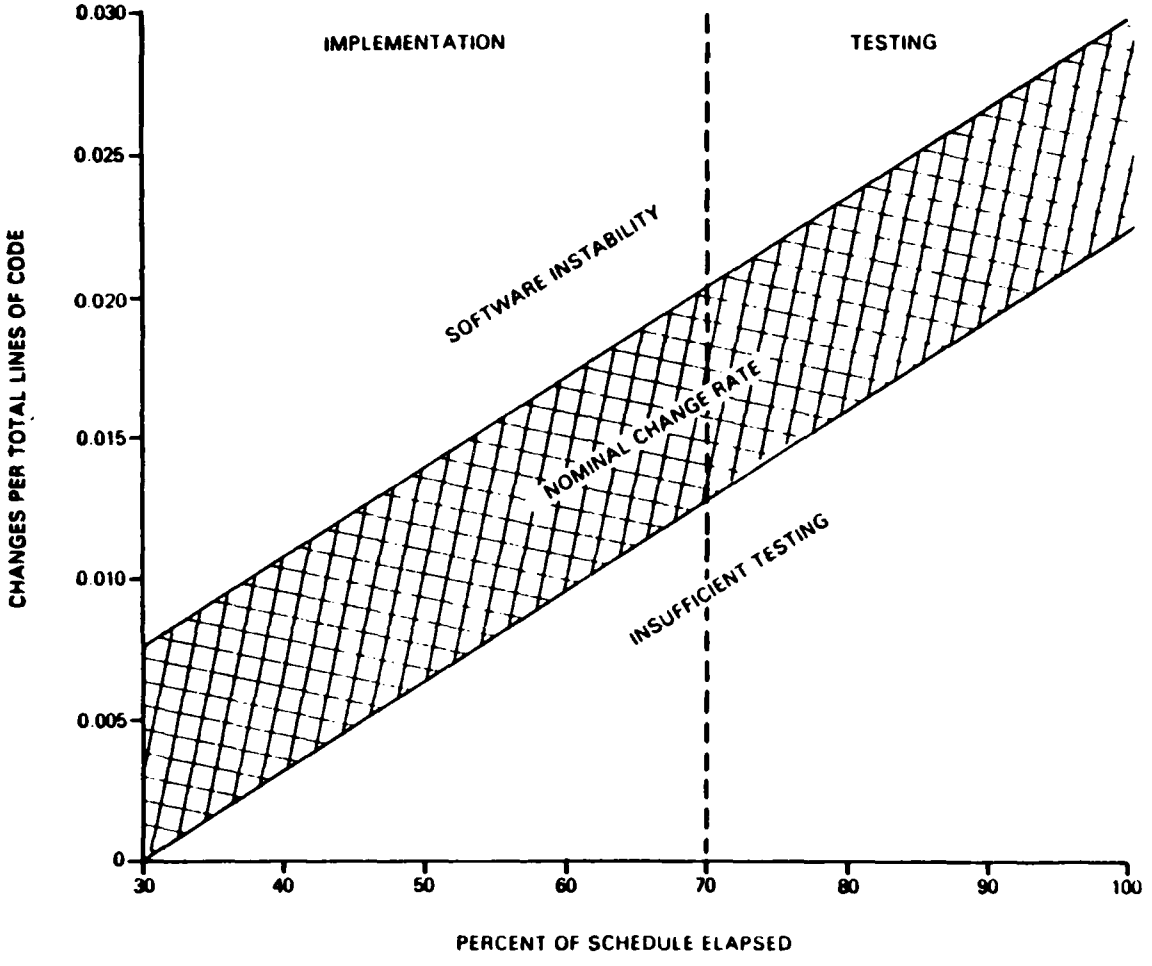


Figure 2-5 Nominal Software Change Pattern

3. METHODOLOGY REQUIREMENTS FOR SOFTWARE METRICS

The software development process and auditing activities expand from the early phases of the system acquisition and development process to the software operation and maintenance phases. The quality metrics methodology includes four major phases:

- Software Quality Specifications
- Software Quality Estimation
- Software Quality User Evaluation
- Software Metrics Tuning.

The software quality specifications are established early in the development process. Software scores are calculated at predetermined events which imply the application of scoring criteria to deliverables available at that event. The auditing activities during the operation and maintenance phases provide the data necessary for tuning the quality metrics which were applied during the previous development phases. By doing so, the relationship between metrics scores and user perceived quality can be improved and possibly validated. The four phases are described further in the following sections.

3.1 SOFTWARE QUALITY SPECIFICATIONS

This phase starts very early in the life cycle of a product, as soon as a description of the system, although incomplete and informal, is available. Three major objectives must be accomplished in this phase:

a) Determine the criticality of the software

b) Establish the technology risks

c) Specify the software quality criteria and targets.

A cooperative effort is required. Inputs from candidate developers, the System Program Office (SPO), the target user command, the logistic command, and the product division quality assurance department are needed so that all conclusions properly reflect the different needs

and requirements of all parties involved. To change those conclusions during the development program can be very costly.

The criticality of the software is the key parameter for establishing the level and formality of the activities required, as it was previously discussed. It is very important to establish the proper level of criticality because the development cost of software rapidly increases as a function of the criticality. The criticality of the software is a function of the criticality of the application and a function of system design consideration. As an example, if the application has a high level of criticality and no redundancy is available within the system to perform that same function, then the software which implements that function is very critical. On the other hand, if the system provides some form of redundancy to perform the same function, in the case of a software failure, then the software may be assigned a lower level of criticality, which would decrease the development cost.

The technology risks associated with the software development may be an important factor for selecting the proper development process, WF or RP. As it was previously discussed, the RP model, or variations of it like the Spiral model, may be very effective to support the early phases of the development process in the case of high technology risk areas and in the case of complex user interface requirements.

The software quality criteria and targets must be established as a function of the two previously discussed considerations, criticality and risk, and as a function of the other relevant software characteristics which have been previously identified. They are: embedded vs. general purpose; space vs. ground based; life cycle; size and cost; and intended use. For each of those characteristics, quality factors which are typically very relevant have been identified. It must be realized, however, that any product may require quality factors different from those identified, because of special requirements or considerations. As an example, portability is a quality factor identified for products with long life cycle. Some products with short life cycle, however, may also be required to be portable. The ultimate objective of a software metrics science is to be able to precisely correlate numerical scores of metrics, calculated during the development process, to quantifiable characteristics of the final product. As an example a maintainability score of .9 would correspond to an average effort to detect and correct software errors not greater than 1 man-day. The current technology, however, is far from being capable of supporting such a worthwhile

objective. As one example of the problems involved, consider attempting to balance conflicting quality requirements, where high efficiency is required. The process of balancing requirements implies: a) the knowledge of the quantitative relationships between metrics scores and quality ratings of the final product, so that the proper targets for metric scores can be established; b) the knowledge of boundaries of feasibility in the multidimensional space of quality targets, so that among all combinations of target values, feasible and unfeasible combinations are identified; and c) the selection of the feasible combination which best approximates the combination of target values for a specific software product.

The technology for performing the three tasks above, however, is not currently available. Furthermore, neither Government nor industry has squarely addressed and quantitatively stated the required compromises to date. We strongly recommend the initiation of a program to address these important technology requirements. Until that is complete, only qualitative and approximate judgment is available for supporting the difficult process of balancing competing quality factors.

Considerable compromises and initial steps must be taken first. We propose the following approach for each software product with unique combinations of characteristics:

1) Determine the software criticality level (1, 2, or 3);

2) Identify the quality factors which are most critical, and assign minimum acceptable scores to each of them;

3) Identify the quality factors which are less critical, and assign a minimum acceptable score to each of them. Scores in this category are expected to be lower than the scores in the critical category;

4) Scores of each quality factor must be evaluated at the end of each relevant development activity.

Scores must met or exceed the minimum acceptable values. The criteria of setting scores are, at least initially, rather arbitrary because of lack of validated metrics models. As more knowledge is available, stable correlations can be established between scores and final product characteristics. Then the scores can be assigned in a manner which reflect the desired characteristics of the final product.

The importance of establishing the software criticality is that the criticality determines, to a large extent, the formality and the extent of the entire development effort, including all audit, review and quality metrics activities of the development effort.

The targets are defined during the earliest phases of the system development process, and they are further refined at the end of the software requirement analysis. The metrics are then applied, for monitoring purposes, throughout the software life cycle at events such as the end of preliminary design, detail design, unit coding, and integration phases. Requirements for user feedbacks during the operation and maintenance phases, are also developed so that the metrics can be validated. The most appropriate phases and events for applying metrics are identified which satisfy the auditing requirements of the standards and provide estimates of quality and cost in the most effective manner. The WF and RP approaches are both considered and separately developed. Methods for establishing the level to which the final product meets the requirements are also developed.

The detailed steps of the methodology are:

a) The process starts with the need for developing a software product. Any constraint which affects the development process is identified at this time.

b) The software product to be developed must be assigned to a specific software domain so that quality and productivity requirements can be set.

c) The metrics methodology is defined so that metrics scope and phases are defined. The application of the metrics continues throughout the development process and it provides estimates of the level to which quality and productivity requirements will be satisfied, once the product is fully developed. At product completion, the best assessment of the quality is made, which takes into account all scores of each metrics applied during the entire development process.

d) After the product is put into operation, the final user will independently re-assess the quality of the product as he perceives it. He will be the final judge of how well the product satisfies the quality and productivity requirements which were established at the earliest phases of the development processes.

e) The inputs of the user will be utilized for tuning the model of the metrics. If good agreement exists between the predicted quality of the product (by the metrics) and the user perceived quality, then the metrics do not need to be further tuned. If disagreement exists between predicted and actual, then some metrics need to be properly modified (tuned). The process of metrics fine tuning extends for several software products until good agreement is consistently achieved.

Figure 3-1 depicts a flow model establishing and applying the software measurement process for a given system development process.

3.2 SOFTWARE DOMAIN

The software domain for the product is defined based on the guidelines described in Section 2 of this document. Quality and productivity targets are then grossly defined for the software product and the development process. Remaining conflicting requirements are outlined and resolved either by eliminating less critical requirements, or by upgrading hardware requirements (which decreases the need to have highly efficient software, a requirement which is in negative correlation with many others), or by further segmenting the software product so that contrasting requirements are eliminated or limited to small portions of the software product only. Alternatively, the relative importance of each quality factor can be weighted and the measurands which are related to a heavily weighted quality factor, prevail, in the case of conflict over those which are related to a lightly weighted quality factor.

Final quality and productivity targets must be negotiated between the user and the developers so that all parties involved realize the impact, the difficulties and the uncertainties.

As an example of how an actual development of SDS software fits into the categories and domains defined, we take the Space-Based Experimental Version (SBEV) prototyping, for which SPARTA is the prime contractor. Under this contract, several prototype systems are being developed to resolve critical issues and investigate the feasibility of various approaches. One of the prototypes being developed is the SDS Communications Module (SCM). The SCM has as its purpose to provide all the interplatform and intraplatform data communication services for SDS applications (such as the Battle Manager). The prototype SCM is being developed in Ada, using the Rapid Prototyping software development model. It is expected to be

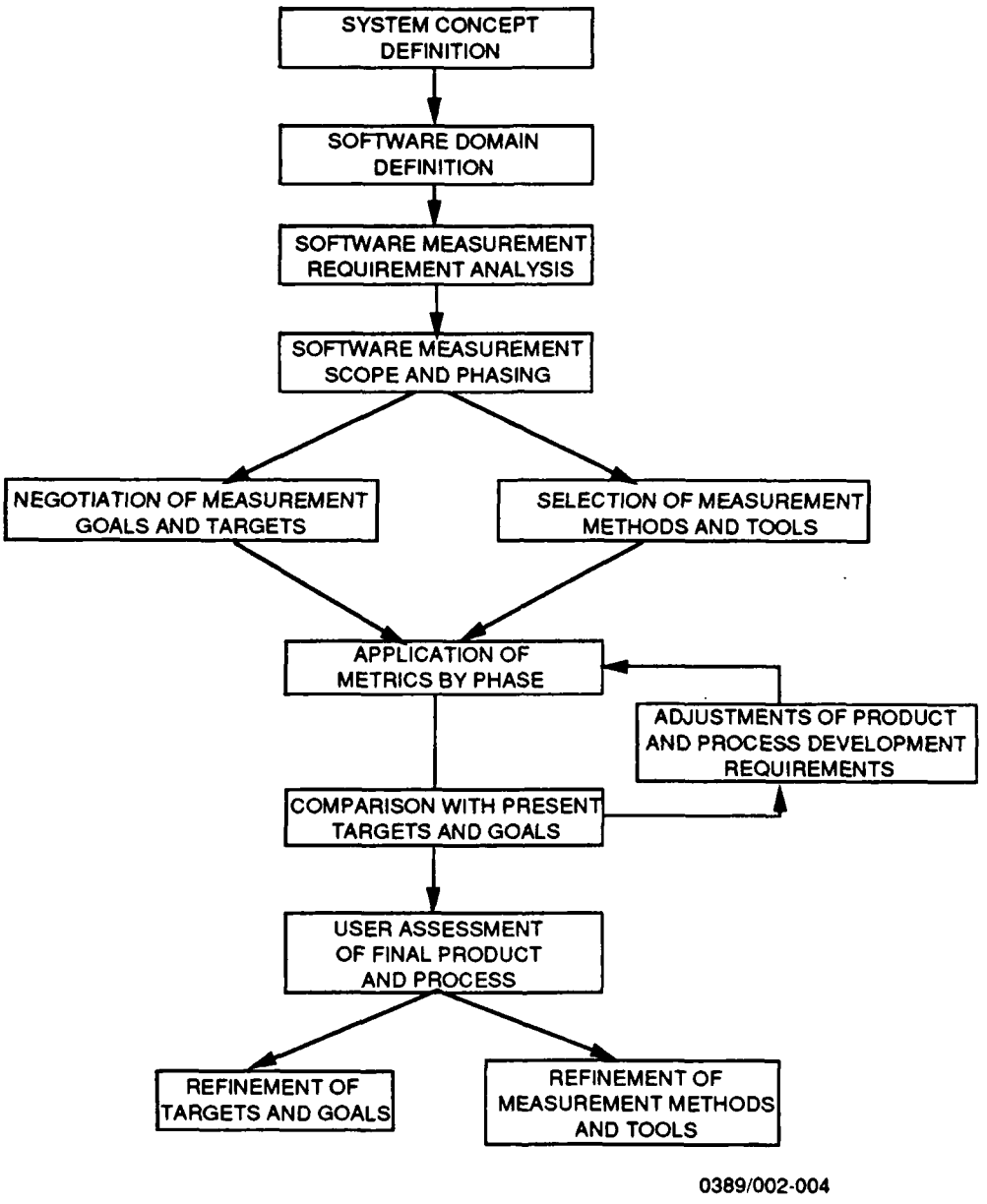


Figure 3-1 Flow Model for Establishing and Applying the Software Measurement Process

comprised of approximately 12000 lines of Ada code. Both the development systems and target systems are Sun workstations.

From the set of application software domains described in Section 2.2, the Prototype SCM development effort best fits into the domain of Demonstration Simulation. From Section 2.4.30, the software characteristics typically associated with that domain are:

Time and Function Critical;

Embedded and Near Real-Time;

Ground-Based;

Life Cycle Through Development;

Algorithmic Intensive;

Large;

Moderate Technology Risk;

Non-Operational Use.

From Section 2.4, the SCM Prototype Development is then Process Typed. From the descriptions of these levels, the SCM Prototype Development is best classified as Level 2. While not large in expected development, it is critical that the salient data communications functions be accurately represented ***(the eventual goal is to provide an emulation, rather than a simulation).***

At this point we combine the software characteristics with the software level, as we specifically evaluate the unique properties of the SCM effort. The following restatement (with modifications) of the software characteristics are:

Time and Function Moderately Critical;

Embedded;

Ground-Based;

Life Cycle Through Development;

Low/Medium Algorithm Intensive;

Small;

Moderate Technology Risk;

Non-Operational.

We now select the appropriate quality factors associated with the above characteristics. For this development effort, they are:

Reliability:	Medium
Survivability:	Low
Integrity:	High
Efficiency:	Medium
Throughput:	High
Maintainability:	Medium
Usability:	Low
Portability:	Medium
Re-Use:	Low

From the above list, it is apparent that software measures that address Integrity and Throughput would rank first, followed by *Reliability*, *Efficiency*, Maintainability and Portability, with Survivability, Usability, and Re-Use being the least important.

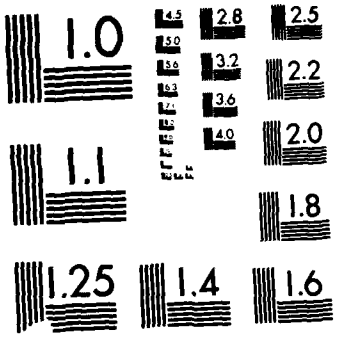
3.3 METRICS SCOPE AND PHASES

A most important task is to define the criticality of the software product. If the product is assigned a level of criticality which is excessively high, then the development costs are also unduly high. If the product is assigned a level of criticality which is excessively low, then it will not be developed to the level of quality which is needed. Software criticality is a function of many system level attributes like: criticality of the function to be performed; availability of functional, analytical, or hardware redundancy (such availability decreases the level of criticality of the software). Critical software requires more frequent and stringent audits than non critical software. Criteria are defined for establishing two or three levels of criticality of the software.

Other important factors are size, complexity and cost. Large and costly programs require more frequent and stringent audits than small programs. Criteria must be defined for establishing the optimum monitoring requirements as a function of size, complexity and cost. The selection of the criteria and of the metrics elements to be applied to each product and process, is made based on analytical evaluation and published data relative to the effectiveness of each metric.

AD-A207 449 SDS SOFTWARE MEASUREMENT REQUIREMENTS(U) ANALYTIC SCIENCES CORP ARLINGTON VA 06 APR 89 TASC-TR-9033-1 SDIO84-88-C-0018 2/2

UNCLASSIFIED F/G 12/5 NL



UTION TEST CHART

The SDS software can greatly benefit, in productivity and quality, if software reuse is encouraged and rewarded. A metric to monitor software reuse, throughout the development process, needs to be developed so that quantitative and objective measures of the percentage of reuse in each product can be assessed. It is important to notice that software reuse is not limited to code, but it includes requirements specifications, design, documentation, process, interfaces (including user interfaces) and others.

3.4 USER FEEDBACKS

A methodology is necessary for formalizing the user feedback, relative to the quality of the product, once the system is in operation. The user, and his judgement, are the only available feedback of the quality of the software. Because user judgement can vary from time to time and clearly from user to user, a methodology must be established to ensure that consistent judgements are supplied every time. That is, the rigor of the user judgement must be of the same level of formality used for scoring the metrics. For instance, if the user has to judge the level of maintainability of a product, first he must use the same scale that was used for the requirements (high, medium, low for example), and then he must be given exact formulas for evaluating the product. Then every user will utilize the same formula for every product, every time. The determination of such formulas is beyond the scope of this document.

3.5 METRICS TUNING

The metrics criteria, elements, and measurands are critically reviewed, based on the correlation between the predicted quality, based on the metrics, and the perceived quality, based on the user inputs. If the two values are in agreement, no tuning is required of the metrics. If considerable disagreement exists between estimated and user perceived quality factors, then further tuning of the models is required, so that the correlation can be improved for future products.

END

FILMED

6-89

DTIC