

AD-A163 756 SOFTWARE REPORTING METRICS REVISION 2(U) MITRE CORP
BEDFORD MA R J COLES ET AL. NOV 85 NTR-9650-REV-2
ESD-TR-85-145 F19628-86-C-0001

SOFTWARE REPORTING METRICS REVISION 2(U) NITRE CORP
BEDFORD MA R J COLES ET AL. NOV 85 NTR-9650-REV-2
ESD-TR-85-145 F19628-86-C-0001

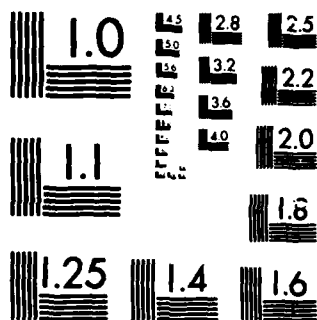
1/1

UNCLASSIFIED

F/G 9/2

NL

[illegible]



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

ESD-TR-85-145

MTR 9650
Revision 2

11

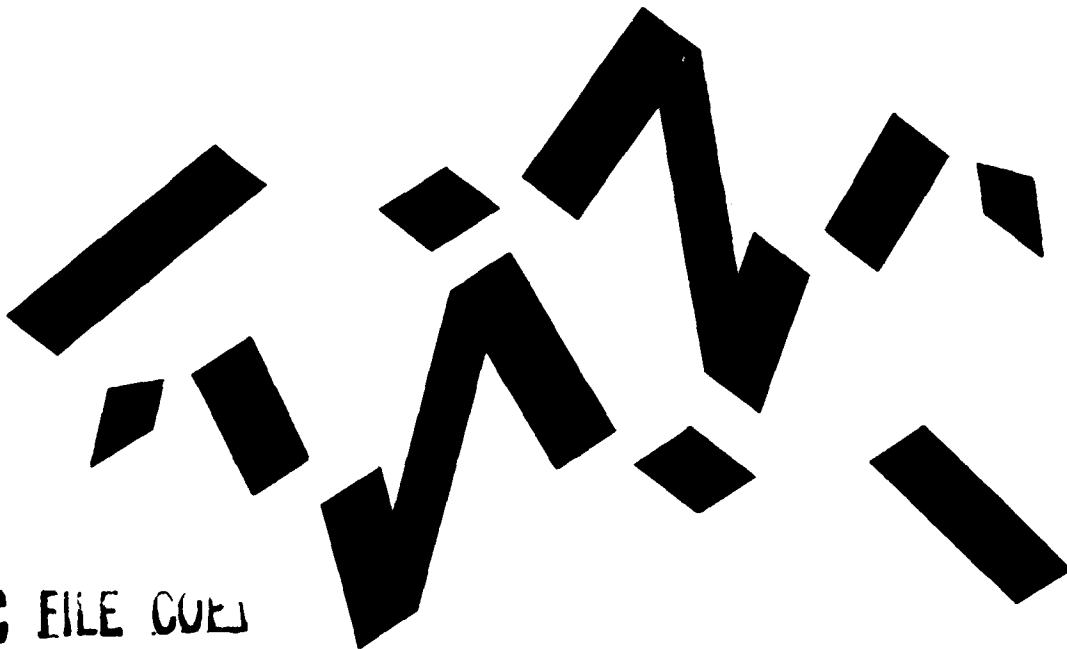
AD-A163 756

Software Reporting Metrics

November 1985



DTIC
ELECTE
FEB 10 1986
S D
P D



DTIC FILE COPY

Prepared for Deputy for Acquisition Logistics and Technical
Operations, Electronic Systems Division, AFSC, United States
Air Force, Hanscom Air Force Base, Massachusetts

Approved for public release.
distribution unlimited

NTRE



86 2 7 313

When U.S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.



GEORGE G. JACKELEN, Major, USAF
Project Officer, Project 5720
Computer Technology and Support Division

FOR THE COMMANDER



ROBERT J. KENT
Director, Computer Systems Engineering
Deputy for Acquisition Logistics
and Technical Operations

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified			1b. RESTRICTIVE MARKINGS														
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.														
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE																	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MTR-9650, Rev. 2 ESD-TR-85-145			5. MONITORING ORGANIZATION REPORT NUMBER(S)														
6a. NAME OF PERFORMING ORGANIZATION The MITRE Corporation		6b. OFFICE SYMBOL (If applicable)		7a. NAME OF MONITORING ORGANIZATION													
6c. ADDRESS (City, State and ZIP Code) Burlington Road Bedford, MA 01730		7b. ADDRESS (City, State and ZIP Code)															
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Deputy for Acquisition (cont.)		8b. OFFICE SYMBOL (If applicable) ALSC		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F19628-86-C-0001													
8c. ADDRESS (City, State and ZIP Code) Electronic Systems Division, AFSC Hanscom AFB, MA 01731-5000		10. SOURCE OF FUNDING NOS. <table border="1"><tr><td>PROGRAM ELEMENT NO</td><td>PROJECT NO</td><td>TASK NO.</td><td>WORK UNIT NO.</td></tr><tr><td></td><td>572H</td><td></td><td></td></tr></table>				PROGRAM ELEMENT NO	PROJECT NO	TASK NO.	WORK UNIT NO.		572H						
PROGRAM ELEMENT NO	PROJECT NO	TASK NO.	WORK UNIT NO.														
	572H																
11. TITLE (Include Security Classification) SOFTWARE REPORTING METRICS																	
12. PERSONAL AUTHOR(S) Coles, R.J.; Kasputys, J.A.; Lasko, K.L.; Saunders, T.F.; Schultz, H.P.																	
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM TO		14. DATE OF REPORT (Yr., Mo., Day) 1985 November 1985													
				15. PAGE COUNT 60													
16. SUPPLEMENTARY NOTATION																	
17. COSATI CODES <table border="1"><tr><td>FIELD</td><td>GROUP</td><td>SUB. GR</td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>			FIELD	GROUP	SUB. GR										18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Project Management Software Acquisition Status Monitoring		
FIELD	GROUP	SUB. GR															
19. ABSTRACT (Continue on reverse if necessary and identify by block number) This paper presents a set of software management metrics that have been developed by the ESD/MITRE Software Center. The metrics are designed to assist in monitoring the progress of software development by gaining better visibility into the software acquisition process. This revision reflects government and industry comments on the previous version, and includes new charts, information on application of the metrics, and appendices containing generic and sample data items.																	
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☒ DTIC USERS ☐			21. ABSTRACT SECURITY CLASSIFICATION Unclassified														
22a. NAME OF RESPONSIBLE INDIVIDUAL Diana F. Arimento		22b. TELEPHONE NUMBER (Include Area Code) (617)271-7454		22c. OFFICE SYMBOL Mail Stop													

DD FORM 1473, 83 APR

EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

8a. Logistics and Technical Operations

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

ACKNOWLEDGMENTS

This report was prepared by the MITRE Corporation. The work was sponsored by the Computer Technology and Support Division (ALSC), Deputy for Acquisition Logistics and Technical Operations of the Electronic Systems Division (ESD) of the United States Air Force, Hanscom AFB, MA 01731. Funding for this report was provided by Project 5720, ESD/MITRE Software Center General Support. This project is the ESD-initiated effort to improve the acquisition of Mission-Critical Computer Resources (MCCR). The goals of the project are to: provide guidance, tools, systems, and techniques to Program Offices; interact with Air Force and DOD organizations that establish policies, regulations, and standards for software acquisition; and direct associated technology efforts.

This report was developed through the cooperative efforts of the ESD/MITRE Software Center staff with supporting commentary from staff throughout the MITRE Corporation and ESD. Some of the data has been obtained from Dr. Barry Boehm's Software Engineering Economics (Ref. 1). Thanks are also given to the National Security Industrial Association (NSIA) who, at the request of Lt. Gen. Melvin F. Chubb, Commander of ESD, provided a working group that actively reviewed and commented on the previous version of this document. The generic data items in Appendix B were developed in part from information provided by ESD/ALSC. This document has also been revised for compatibility with the new Defense System Software Development Standard, DOD-STD-2167.

TABLE OF CONTENTS

<u>Section</u>	<u>Page</u>
LIST OF FIGURES	vi
LIST OF TABLES	vi
1 INTRODUCTION	1
2 SOFTWARE REPORTING METRICS	3
3 APPLICATION OF THE METRICS	35
4 CONTINUING METRICS EFFORTS	41
REFERENCES	45
 <u>Appendix</u>	
A GENERAL INFORMATION FOR SOFTWARE DEVELOPMENT	47
B GENERIC METRIC DATA ITEMS	51
C ENHANCED METRIC DATA ITEMS	55
D DATA DEFINITIONS	61



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

LIST OF FIGURES

<u>Figure</u>		<u>Page</u>
1	Software Size	5
2	Software Personnel	9
3	Software Complexity	13
4	Development Progress	17
5	Testing Progress	21
6	Computer Resource Utilization Estimates	25
7	Software Volatility	29
8	Incremental Release Content	33

LIST OF TABLES

<u>Table</u>		<u>Page</u>
1	Software Complexity Ratings	15

SECTION I

INTRODUCTION

This document is intended to help Project Managers prepare software development status reports. The document describes Software Reporting Metrics that may be generally applied to check the progress of software development in an acquisition project. It also provides information that reflects experience on previous acquisition projects.

System Program Offices (SPOs) are expected to use these metrics to augment conventional acquisition system development reports such as cost and schedule reports. The successful use of these metrics depends on the program manager's enforcement of a serious technical review of the monthly numbers. The metrics provide a top level overview of software development status. They are not intended to provide an in-depth evaluation of software. It is only through reviews and related interpersonal communications that the significance and relevance of the contractor's estimates can be ascertained.

Section II ~~of the document~~ presents each metric. A description is provided of the information intended to be evaluated, the normal behavior patterns that may be expected, and the mechanics of collecting and reporting the metric data. Each metric is accompanied by a sample plot. The plot is intended to illustrate representative data for a 24-month acquisition effort. An explanation of the possible interpretation of the plot is presented. Also, a list of notes is included that provides some general guidance in interpreting the data.

Section III discusses application of the metrics. An explanation is presented of how the metrics can be expanded in scope and depth to address project specific needs. Also discussed is why the metrics data should not be the only parameters collected; they should be augmented by more detailed reports appropriate to the specific project. This section also discusses the mechanics for obtaining and processing the metrics data, and the pros and cons of using electronic transfer as a means of data collection. A brief discussion of an automated tool for storing, processing, and presenting the metrics is also included.

Section IV ~~of the document~~ describes continuing work efforts in the area of software status monitoring and metrics. This section includes areas that will be addressed and integrated with the metrics efforts. The immediate thrust of this integration effort is in the area of cost and schedule data. In addition, many comments have been received on the metrics, and there are plans to address and incorporate those comments not reflected in this revision.

Appendix A of the document presents general information and guidelines that reflect past experience with software acquisition. The reader is cautioned that this information is intended to be broadly applicable to the type of Command, Control, Communications, and Intelligence systems that are acquired by the Air Force and other DOD organizations. Its broad nature means that it is inherently NOT ABSOLUTE. Cases exist that legitimately violate the limits indicated; but, caution is recommended whenever a project proposes to violate any of these factors.

Appendices B and C present generic and sample data items, respectively, for collecting the necessary data to support each software reporting metric. Appendix D contains definitions to assist in understanding these data items.

SECTION II

SOFTWARE REPORTING METRICS

The format of this section has been selected to present each metric on a series of pages. The first page describes the intent of the metric, some comments related to the profile presented when data is collected and plotted, and a definition of the data to be collected. The "definition" is intentionally loose. The definition of the data to be collected should be formally established through Data Item Descriptions (DIDs) or Program Management Review (PMR) deliverables that are tailored to each software acquisition program.

The second page of each metric contains a sample chart of metric data and a brief discussion interpreting the chart. The third page contains notes about the metric and its use. The notes and discussions presented are intended to help in the interpretation of metric data. They should be viewed as general information, not as inflexible principles.

The metrics are plotted on a chart that shows the current month plus the previous 12 months of activity and allows room for the next five months of data to be predicted. Milestones, such as System Requirements Review (SRR), System Design Review (SDR), Software Specification Review (SSR), Preliminary Design Review (PDR), Critical Design Review (CDR), Test Readiness Review (TRR), Functional Configuration Audit (FCA), Physical Configuration Audit (PCA), and Formal Qualification Review (FQR) may be indicated on the abscissa. In some cases, planned values are plotted. Where planned data is shown, it is the data derived from the first plan submitted to the Government. For example, the personnel profile is usually provided in the proposal, so the personnel profile from the contract is shown. An additional planned profile may be included if the contractor is reporting his current anticipated profile. Another example is in testing where the planned profile should be that established when the contractor's test plan was approved.

The sample plots shown in this document are formulated from a hypothetical project. This hypothetical project was identified by the contractor to require a software development of 120 thousand Source Lines of Code (SLOC) and 1200 staff months over a 24-month schedule. The project milestones are as follows: SRR at month one, SDR at month two, SSR at month three, PDR at month seven, CDR at month 12, TRR at month 20, and FQR at month 24. In each of the sample plots the current month is identified.

1. SOFTWARE SIZE

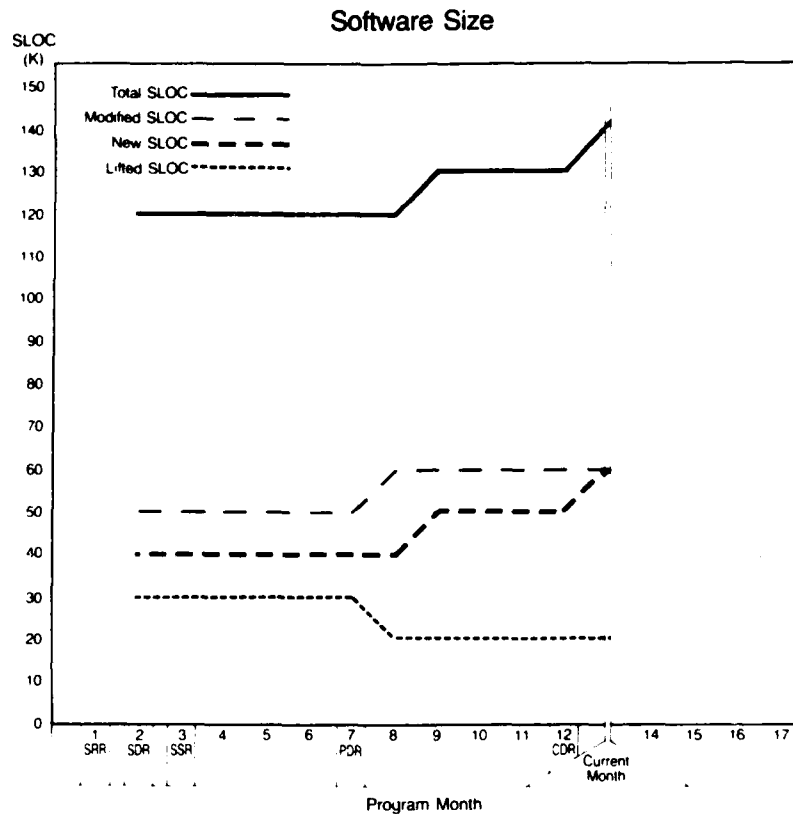
The Software Size indicators are intended to show the magnitude of the software development effort. Source Lines of Code (SLOC) affect the software engineering effort necessary to build the system. Growth in SLOC can lead to schedule slips and management problems due to understaffing.

Most programs exhibit rising estimates of SLOC over time. The development effort for a system may be reduced if some of the SLOC can be reused from other programs. There is some effort associated with integrating lifted code (that is, borrowed from some other source without change, such as, operating systems or an application previously completed) and even more effort when modifications are needed. The degree of effort must be estimated for each system.

The count of Total SLOC, SLOC to be lifted from other programs, SLOC to be modified from other programs, and SLOC to be developed as new code for this program should be updated and reported at the end of each calendar month of the contract. The term "SLOC" is often interpreted to exclude nondelivered support software such as test drivers. However, with respect to these metrics, newly developed support software programs should be included in the Software Size counts.

Any reasonable definition of SLOC may be used as long as both the contractor and the SPO understand and accept it. An example of a definition of source instructions, found in Reference 1, is presented below.

SLOC includes all software instructions created by project personnel and processed into machine code by some combination of preprocessors, compilers, and assemblers. It excludes comment cards and unmodified utility software. It includes job control language, format statements, and data declarations. Instructions are defined as lines of code or card images. Thus, a line containing two or more source statements counts as one instruction; a five-line data declaration counts as five instructions.



DISCUSSION

At PDR for the hypothetical project, the chart shows the number of lifted lines of code decreasing and the number of modified lines of code increasing by the same amount. Discussions with the contractor revealed that a portion of the code that was originally to be lifted would require modification.

The contractor also explained that the increases in the number of new lines of code at months nine and 13 were the result of requirement changes originating from ECPs.

NOTES

1. The estimated instruction counts for one month should not vary from the previous report by more than 10 percent without explanation. However, continuous growth of even 10 percent would be too high over the life of the program. Typically, growth would be event driven, that is, after PDR and CDR. Commonly, changes will be small between major reviews and potentially large at major reviews. A change in the estimated SLOC count does not necessarily imply trouble -- it may imply that the contractor has a new understanding of the requirements and his design approach. If any SLOC count changes by more than 50 percent, the software development management approach should be reviewed for adequacy.
2. SLOC is a factor that directly affects the number of staff months required for software development. Implementation should be done with a High Order Language (HOL) wherever feasible. Among the benefits of HOLs are increased code reusability, improved maintainability, testability, etc.

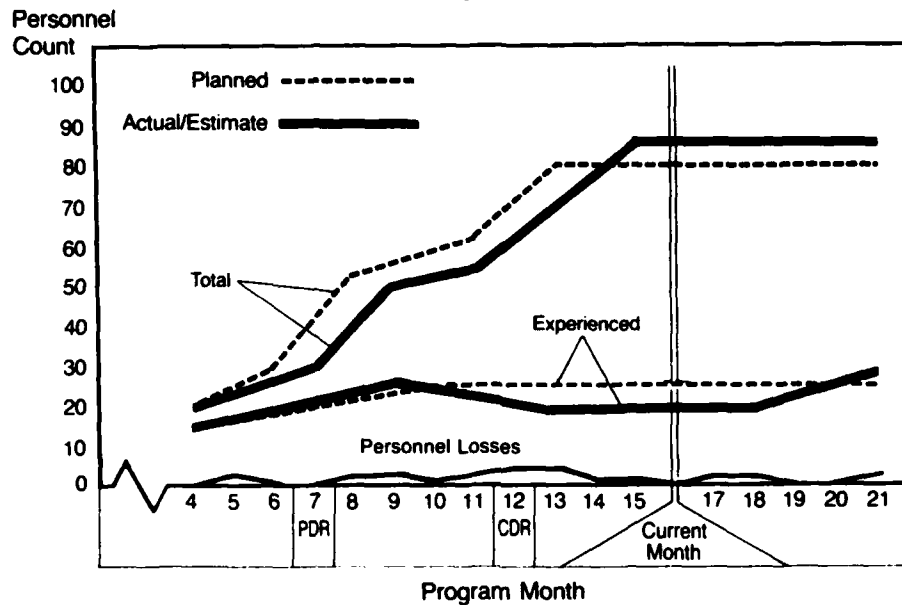
2. SOFTWARE PERSONNEL

The Software Personnel indicators are intended to show the ability of the contractor to apply resources to the program and maintain sufficient staffing for completion of the program. The software staff includes the engineering and management personnel directly involved with software system planning, requirements definition, design, coding, test, documentation, configuration management, and quality assurance. Experienced personnel are defined as those individuals with a minimum of three years experience in software development for applications similar to the system under development.

The staffing profiles for total software staff and for experienced software staff should be plotted at the beginning of the contract. A normal program may have some deviations from the plan, but the deviations should not be severe. A program with too few experienced software personnel, or one which attempts to bring many personnel onboard during the last stages of the project's schedule, is in trouble. The normal shape of the total software staff profile is to start at a moderate level, grow through the design phases, peak through the coding and testing phases, and then gradually taper down as integration tests are successfully completed. The normal shape of the experienced staff profile is to be high at the initial stage of the project, dip slightly through coding and then grow somewhat through testing.

The count of actual total software staff and experienced software staff should be reported at the end of each calendar month. Counts of project personnel lost should be maintained so that the personnel turnover rate can be tracked.

Software Personnel



DISCUSSION

Prior to CDR, the chart shows that the total number of actual personnel is lagging behind the total number of planned personnel. Also, the number of actual experienced personnel is higher than originally planned.

The SPO initiated a dialog with the contractor to discuss the deviations in scheduling. They found that the contractor was having difficulty in staffing and was adding more experienced personnel in an attempt to compensate. The SPO should monitor this closely to ensure that the new plan is adequate and can be accomplished.

NOTES

1. The ratio of total to experienced personnel should never exceed 6:1. (3:1 is a typical ratio for most real time C3 systems).
2. The time required for software development depends on the staff-months delivered.
 - a) Understaffing is an early indication of schedule slippage and potentially an ever-accelerating rate of slippage.
 - b) Adding staff to a late project will seldom repair the schedule.
3. A program that is maintaining the staffing profile, but which is experiencing a high personnel turnover rate is not maintaining needed continuity among the design and implementation staff. The planned rotation of some staff, as the phases of the development change, is reasonable; however, losses that would impair the project knowledge and experience base should be flagged.

3. SOFTWARE COMPLEXITY

The Software Complexity indicator is intended to show the degree of sophistication expected in the software. It does not include development constraints such as schedule and is, therefore, not a full measure of software development difficulty. The complexity indicator is calculated to be an overall indicator. This overall complexity indicator is intended to be derivable from the information used to estimate complexity in B. Boehm's Constructive Cost Model (COCOMO). (See Ref.1.)

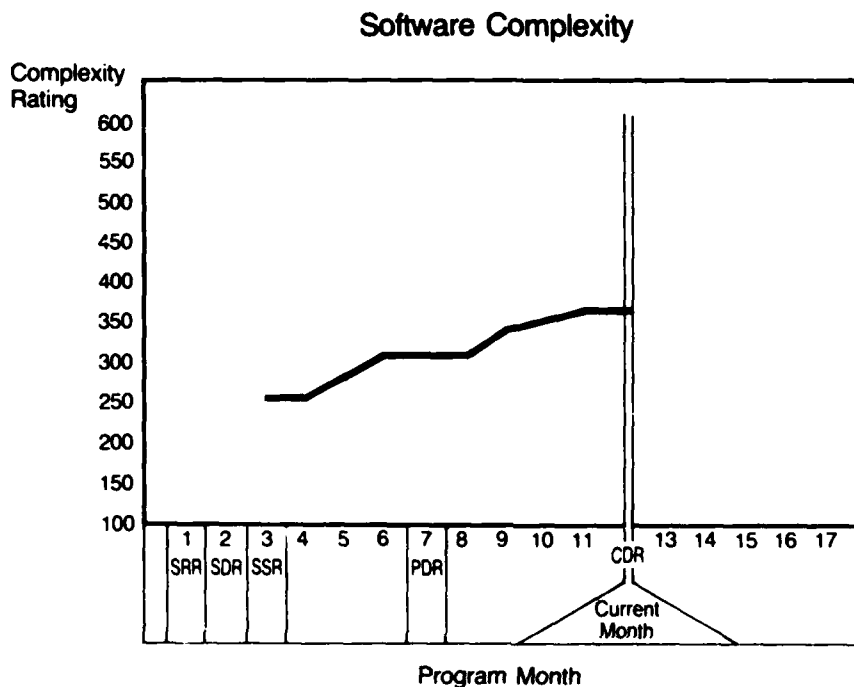
The complexity of the software is not expected to change much through the life of a program, but in some instances change will occur. Changes may occur in response to requirements changes or a reallocation of software functionality among processor resources. It may also change in response to knowledge gained as the design is developed. Changes in the complexity rating should be interpreted as an indication of important changes in the program.

The Software Complexity indicator is calculated by combining the weighted average of the complexity ratings for each Computer Software Configuration Item (CSCI) and all non-delivered (but developed for this program) support software. Weighting is established according to the percent of the total software (SLOC count) contained in each CSCI or necessary Support Software Programs (SSP). Note that lifted or modified code that may be included in the CSCIs will be included in the calculation.

The ratings in Table 1 will be used for determining complexity. A CSCI's complexity should be the highest complexity rating appropriate for the CSCI's commonly performed operations. The percent of the total software contained in each CSCI or SSP will be multiplied by:

- 1 for a "very low" complexity rating;
- 2 for a "low" complexity rating;
- 3 for a "nominal" complexity rating;
- 4 for a "high" complexity rating;
- 5 for a "very high" complexity rating; and
- 6 for an "extra high" complexity rating.

The total of the weighted complexity ratings will result in a number between 100 and 600. This number will be updated and reported at the end of each calendar month of the contract.



DISCUSSION

In the chart, the complexity rating increased during preliminary design and during detailed design.

During preliminary design, the SPO was concerned over the steady increases and queried the contractor. They found that the increases during preliminary design were a result of the contractor's increasing awareness of the complexity of the software development.

During detailed design, steady increases in complexity alerted the SPO again. After dialogue with the contractor, it was determined that increased complexity was introduced into the system through requirements changes originated by ECPs.

NOTES

1. The complexity rating for one month should not change from the preceding report by more than 10 percent without explanation.
2. Programs with complexity ratings over 450 should have more than 40 percent experienced software staff.
3. Wherever possible, employ mature off-the-shelf operating systems, compilers, database management systems, and support software.
4. For programs of similar size, real-time applications are generally more complex than non-real-time applications.

Table 1. Software Complexity Ratings

Rating	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations
Very low	Straightline code with a few non-nested SP* operators: DOs, CASEs, IFTHENELSEs. Simple predicates	Evaluation of simple expressions: e.g., $A = B + C$ * $(D - E)$	Simple read, write statements with simple formats	Simple arrays in main memory
Low	Straightforward nesting of SP operators. Mostly simple predicates	Evaluation of moderate-level expressions, e.g., $D = \text{SQRT}(B^2 - 4 \cdot A \cdot C)$	No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level. No cognizance of overlap	Single file subsetting with no data structure changes, no edits, no intermediate files
Nominal	Mostly simple nesting. Some inter-module control. Decision tables	Use of standard math and statistical routines. Basic matrix/vector operations	I/O processing includes device selection, status checking and error processing	Multi-file input and single file output. Simple structural changes, simple edits
High	Highly nested SP operators with many compound predicates. Queue and stack control. Considerable intermodule control.	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, roundoff concerns	Operations at physical I/O level (physical storage address translations; seeks, reads, etc). Optimized I/O overlap	Special purpose subroutines activated by data stream contents. Complex data restructuring at record level
Very high	Reentrant and recursive coding. Fixed-priority interrupt handling	Difficult but structured N.A.: near-singular matrix equations, partial differential equations	Routines for interrupt diagnosis, servicing, masking. Communication line handling	A generalized, parameter-driven file structuring routine. File building, command processing, search optimization
Extra high	Multiple resource scheduling with dynamically changing priorities. Microcode-level control	Difficult and unstructured N.A.: highly accurate analysis of noisy, stochastic data	Device timing-dependent coding, micro-programmed operations	Highly coupled, dynamic relational structures. Natural language data management

* SP = structured programming

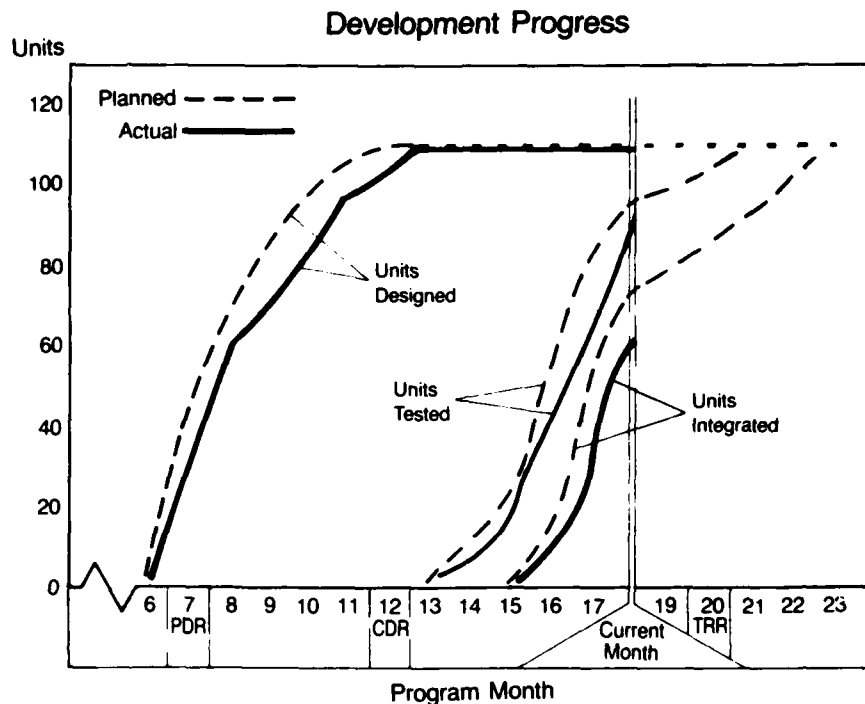
Barry W. Boehm, *Software Engineering Economics*, ©1981, p. 122.
Reprinted by permission of Prentice-Hall, Englewood Cliffs,
New Jersey.

4. DEVELOPMENT PROGRESS

The Development Progress indicator is intended to monitor the contractor's ability to maintain development progress. It will measure the degree to which the contractor can keep unit test and integration on schedule. After CDR, the next milestone that reveals the status of software development is the Test Readiness Review (TRR). This is much too late in a program to recover if significant problems are discovered. An earlier indication of success or problems is the degree of adherence to a planned schedule for unit design, development, and test.

The design of units usually begins around PDR. Data to monitor the design progress is rarely delivered to the government as a contractual obligation. But if it is available, a count should be plotted of the number of unit design packages that have been closed, that is, the design has passed internal review and is considered "frozen" by the contractor's software development methodology. After CDR, when the contractor begins developing units, a healthy program will experience a steady progression of new units. If the units are being developed in spurts, it indicates problems with managing the software. Sporadic unit development can be caused by factors such as over-utilized development machines or under-experienced staff. It results in a high pressure environment in which all the software is due at once. For a normal program, the total number of units that have passed unit test should be continually increasing, and the total number of units that have been integrated should be similarly rising with a constant slope.

The unit test and unit integration schedule should be reported by the contractor at CDR. The number of units whose design packages have been closed, the number of units passing unit test, and the number of units passing integration tests and accepted into the contractor's configuration control system as integrated software, will be updated at the end of each calendar month of the contract.



DISCUSSION

In this chart, the plot of units actually designed follows closely the plot of units planned to be designed until three months after PDR. At this point the actual number begins to deviate significantly from the planned number. This growing lag prompted the SPO to investigate. This led to the conclusion that increased complexity, introduced by ECPs, prolonged the design of the system.

Currently, the number of units actually tested and integrated lags behind the number of units planned to be tested and integrated. The SPO ascertained from the contractor that the lag resulted from the increased lines of code to be developed, tested, and integrated. These slips may indicate that the project schedule cannot be met. The SPO should closely monitor the progress of testing and integration to assess the likelihood of the project schedule not being met.

NOTES

1. Between PDR and FQR, monitor the software components designed, coded, tested, and integrated as indicators of development progress.
2. Units tested and integrated should progress at a reasonable rate and according to plan.
3. The Software Development Plan should take into account the iterative nature of test and integration. (i.e., The test program should begin and continue throughout the software integration activities.)

5. TESTING PROGRESS

Test Readiness Review (TRR) is the earliest milestone at which the Government can officially determine whether or not the desired functions are being provided. The Testing Progress indicators are intended to show the degree to which the contractor's implementation of the design is meeting program requirements. The testing progress indicators also show whether the test program is going to be extended or whether it may be concluded on a reasonable schedule.

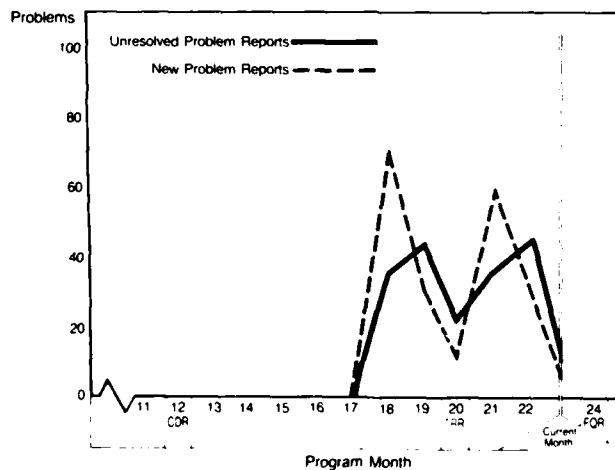
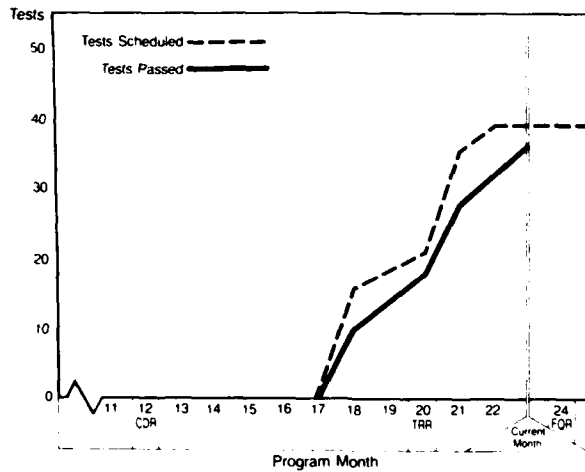
The Tests Scheduled plot should show the cumulative count of tests planned to be completed for TRR and Formal Qualification Review (FQR). A plot of tests that have been successfully passed should overlay the schedule if all tests are passed on schedule. Most programs will experience schedule slips or failed tests. Therefore, the Tests Passed plot can be expected to fall below the Tests Scheduled. The degree to which it falls is an indicator of the readiness of the system to have entered the test program.

Problem reports should be counted. These counts should be plotted on a chart as the incidence of problem reports in the reporting period; additionally, a count of problems remaining open should be plotted. As these problems are resolved, the number of problems remaining unresolved should decrease. Note that an ambitious test plan may prevent the number of unresolved problems from decreasing.

The Testing Progress indicators show a general view of testing progress. It is expected that the SPO will establish a more detailed view by tracking the trouble reports. For example, the longevity of trouble reports can indicate how well the contractor is solving problems. The number of units changed per trouble report can reveal how well functional partitioning has been preserved.

A plot of the cumulative count of tests associated with TRR and FQR should be presented before TRR begins. The count of tests passed, problem reports filed, and unresolved problem reports will be updated at the end of each calendar month of the contract.

Testing Progress



DISCUSSION

In the first chart for Testing Progress, the number of tests passed closely follows the number of tests scheduled.

In the second chart, the number of new problem reports peaked after CSC testing began, then quickly decreased. The same pattern occurred for CSCI testing. The number of unresolved problem reports generally decreased as testing continued.

The program office maintained close contact with the testing staff throughout testing. Although there were problems, they were not major and were resolved quickly.

NOTES

1. The tests passed should converge to the tests scheduled as FQR approaches.
2. The unresolved problem line reveals whether or not the contractor is solving more problems than he is identifying. If the slope of the line is positive (increasing to the right), the test program is revealing problems faster than the contractor can solve them. If the slope of the line is negative, the contractor is on a healthier path towards completing the tests.
3. The problems unresolved should decrease toward zero as FQR approaches.

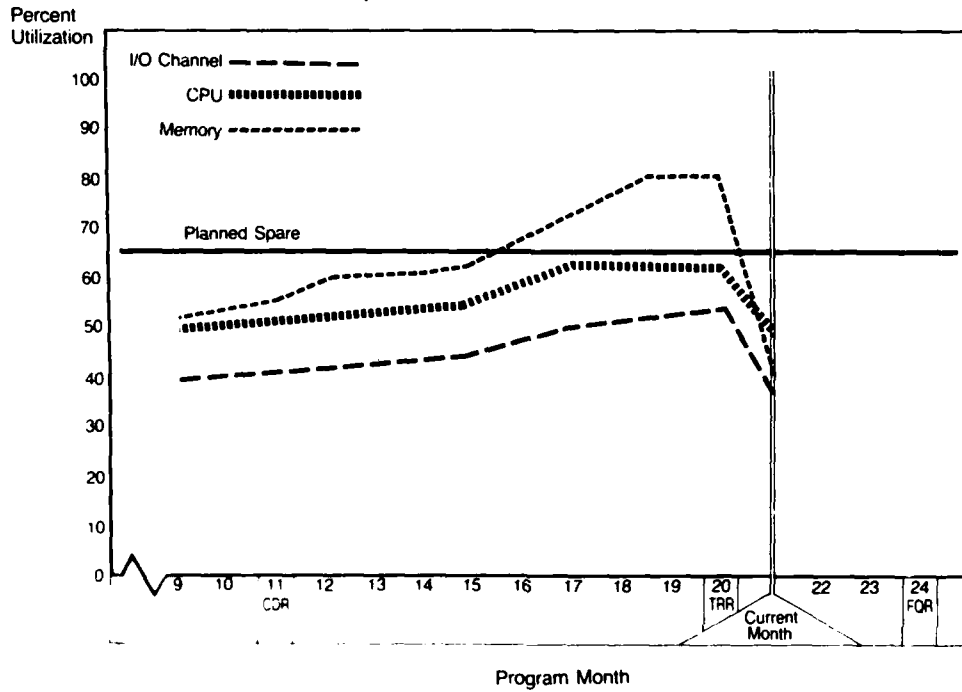
6. COMPUTER RESOURCE UTILIZATION

The Computer Resource Utilization indicators are intended to show the degree to which estimates of the target computer resources used are changing or approaching the limits of resource availability. The three major bottlenecks that can prevent a system from performing within the capacity of its hardware are computation power, memory, and input/output (I/O) channels. Regardless of whether the system architecture is distributed or centralized, these parameters should be carefully monitored to ensure that the software design will fit within the resources planned.

It is normal for large system acquisitions to specify that the completed system will have 50 percent spare capacity in memory, Central Processing Unit (CPU), and peripheral device I/O. This is usually interpreted to mean that the software could require half again as much resource as it uses without exceeding the capacity of the hardware. Most development projects experience an upward creep in the amount of resources estimated to be used. If this upward creep exceeds the 67 percent utilization limit, the project may elect to expand the hardware capabilities. Whenever a resource expansion is approved, the utilization curve, or curves, affected will drop to a new (hopefully below 67 percent) value.

The contractor's estimate for CPU, memory, and I/O resource utilization should be updated at the end of each calendar month of the contract. For projects where multiple computing resources such as a distributed network are used, a separate plot should be developed for each resource.

Computer Resource Utilization Estimates



DISCUSSION

In this chart, each of the resources steadily increased in usage. While CPU and I/O channel usage remained within the planned limit, the memory usage exceeded it. Concern by both the program office and the contractor resulted in additional memory being added to the system.

NOTES

1. CPU utilization should allow a minimum of 50 percent spare.*
2. Planned memory utilization should allow a minimum of 50 percent spare.
3. Planned I/O utilization (channels and data rate) should allow a minimum of 50 percent spare.
4. Performance deteriorates when utilization exceeds 70 percent for real-time applications. Worst case peaks in load cause more frequent conflict among processes competing for processing resources.
5. Resource utilization tends to increase with time. Plan for this expansion early in the software development cycle.
6. Schedule and cost can increase dramatically as the spare drops below 10 percent.

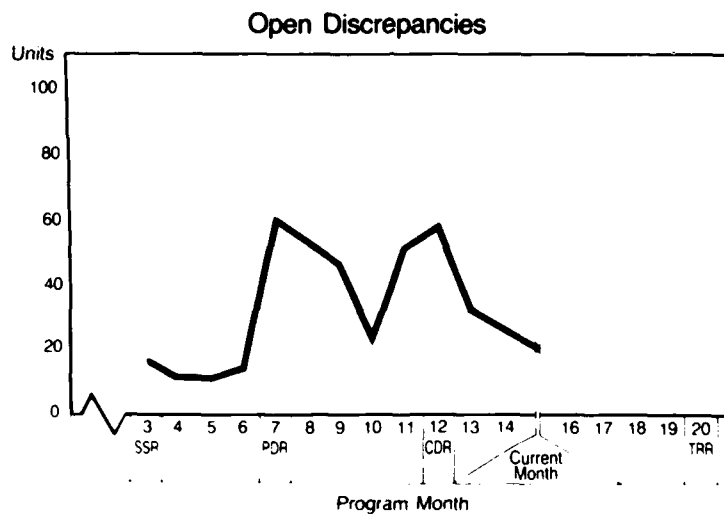
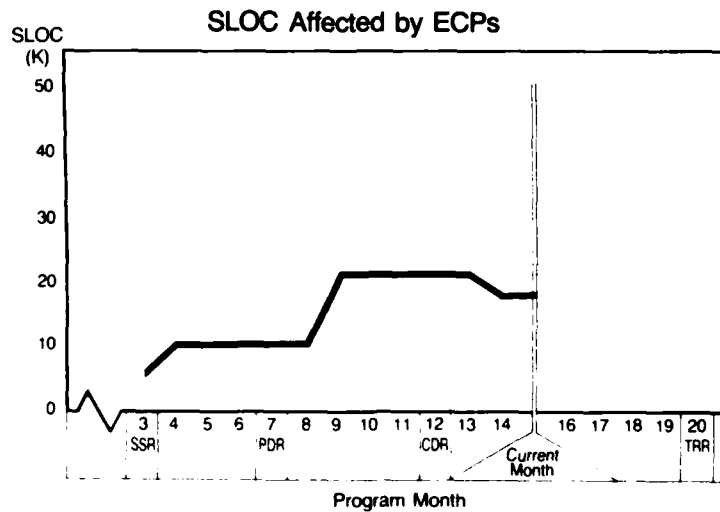
* Confusion exists surrounding the interpretation of "50 percent spare." Contractors often argue that the requirement means they must preserve room for 50 percent growth in their use of a resource. (For example, 67 percent of the resource may be utilized leaving 33 percent [50 percent of 67 percent] as spare.) The Government sometimes uses "50 percent spare" when it really intended to allow only 50 percent of the resource to be used. The government should have requested 100 percent spare.

7. SOFTWARE VOLATILITY

The Software Volatility indicators are intended to show the degree to which changes in requirements or changes in the contractor's understanding of the requirements affect the development effort for a program. When a program is originated, the details of its operation and system design are rarely complete. Consequently, it is normal to experience changes in a system's specifications as requirements are better defined. At some point in the program's history, the requirements should be firm, and only design and implementation issues can cause changes to a specification. These design and implementation issues are usually caught at the Preliminary Design Review (PDR) and Critical Design Review (CDR). When the design reviews reveal inconsistencies a discrepancy report is opened. The discrepancy may be closed (resolved) by modifying or clarifying the design or by modifying the requirements. When a requirement, design, or implementation issue causes a change to the original scope of the project, an Engineering Change Proposal (ECP) may be submitted.

The plot of open discrepancies is expected to spike upward at each review and then exhibit exponentially decreasing behavior. Programs that issue clearly written specifications will experience spikes that are low; programs that have good communications between the program office, systems engineer, and contractor will have a high rate of decay to this curve. For each approved Engineering Change (EC), the software affected should be reported. This indicator will track the degree to which ECs cause an increase in software development effort. Large numbers of ECs and affected SLOC indicate a program that has not established firm requirements before initiating a contract.

The number of discrepancies identified at each review are recorded at the conclusion of the review. A discrepancy is defined as any action item, clarification item, or requirements issue that must be resolved by either the contractor or the Government. The count of open discrepancy reports is maintained by the program office and plotted at the end of each calendar month of the contract. The number of SLOC affected by ECPs is determined by the contractor for each ECP submitted. The cumulative count will be changed only for ECPs that have been approved by the configuration control board. The cumulative count of SLOC affected by ECs will be updated at the end of each calendar month of the contract.



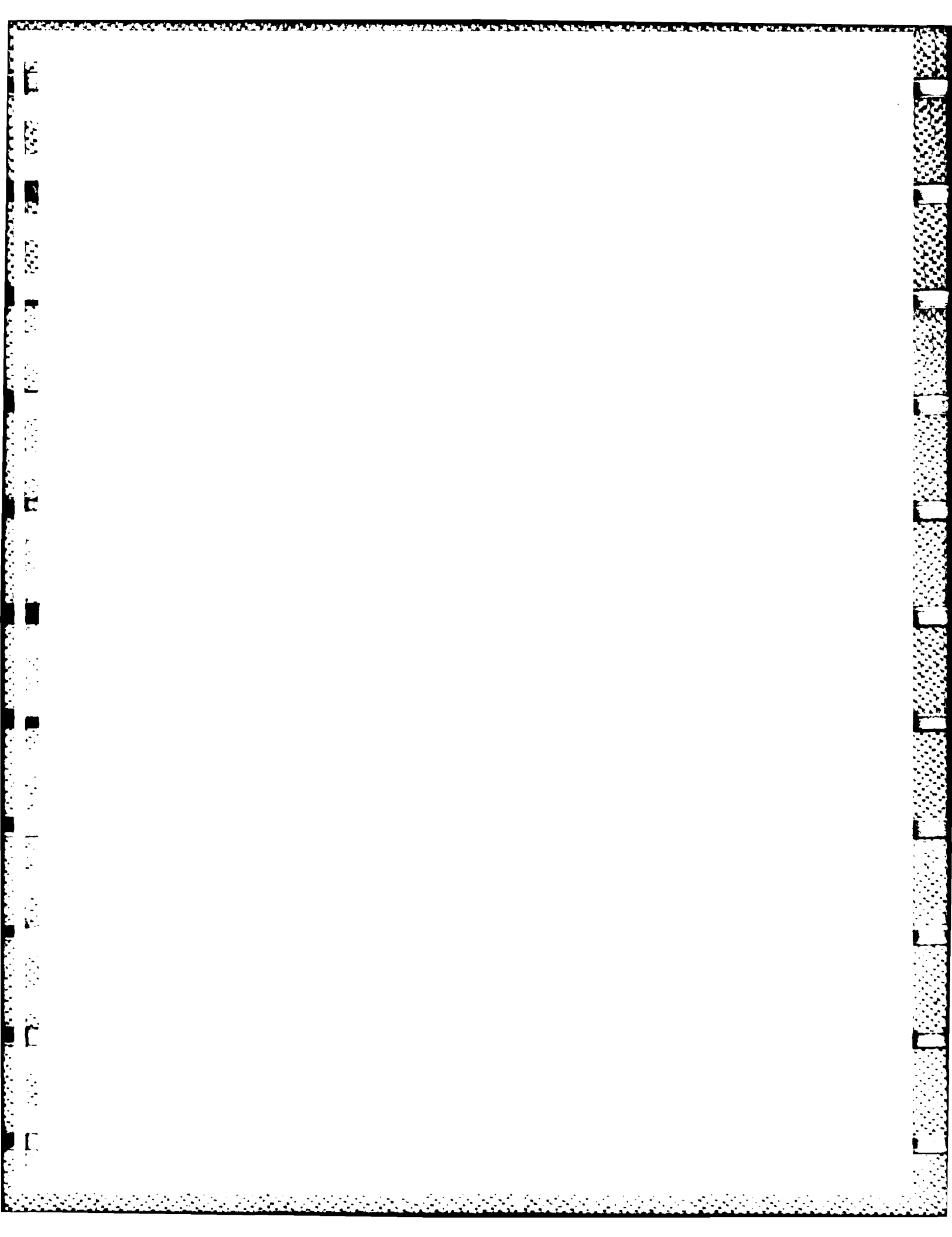
DISCUSSION

Software Volatility has two charts. The first chart shows the the SLOC affected by ECs increasing at months four and nine, then decreasing at month 14. At each increase the SPO determined that the increase was the result of additional requirements. The decrease at month 14 resulted from the relaxation of some requirements.

The second chart shows the number of open discrepancies peaking at PDR and at CDR. The steady decrease in this number after PDR and CDR is attributed to the contractor's ability to resolve the discrepancies.

NOTES

1. Requirement uncertainty leads to changes (ECPs) that result in cost growth and delayed completion of system.
2. Firm requirements should be established before initiating a contract. Alternatively, a planned prototyping approach should be taken.
3. If Software Volatility has not settled down by CDR, the requirements should be frozen for an early delivery increment and the program reopened for requirements review by the users before PDR of the second increment. Note that major requirements issues should have been settled by PDR; if the software volatility remains high to CDR the program is in serious trouble.
4. The count of SLOC affected by an EC should be positive if additional software must be developed to address the EC. The count should be negative if less software is needed and the software in question has not been developed. The count should be unaffected if less software is needed but the software already has been developed.



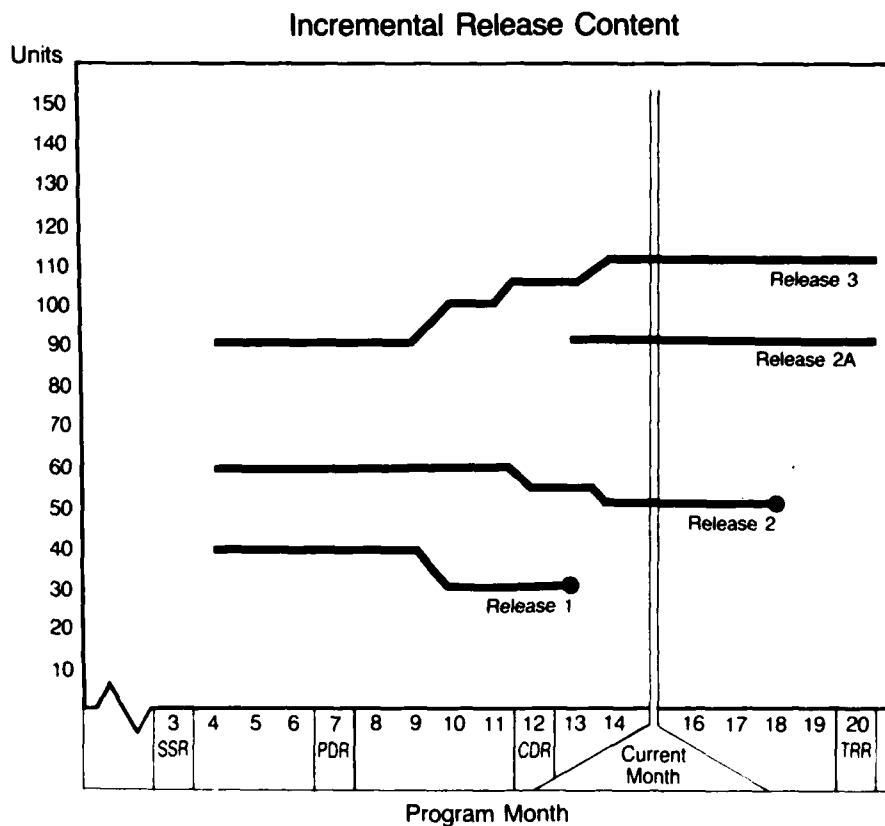
8. INCREMENTAL RELEASE CONTENT

The Incremental Release Content indicator is intended to reveal the contractor's ability to preserve schedule and system function. A common approach used to preserve schedule is to postpone program capabilities. This indicator is useful when the contractor's development plan calls for incremental software builds or "releases." Decreases in the plot of units per release may indicate a program is off-loading functions from early releases to later releases. Increases in the plot of units per release may indicate a program is having unanticipated growth in the complexity of the functions to be delivered.

Proper behavior would be for the number of units to remain constant for each proposed release. Common behavior is for the number of units in early releases to decrease as the release date approaches. In addition, increases in the program's size leads to additional units added to the later releases. In a program whose designs were well prepared for PDR and CDR, the number of units should not increase significantly after CDR. A program that is developing its capabilities on schedule will maintain a constant number of units in each release.

The number of units proposed by the contractor for each anticipated release should be updated at the end of each calendar month of the contract. The plot of unit counts should extend to the completion date for each release.

Note: The term "release" does not necessarily refer to contractually obligated delivery of software. The normal software development process in which the contractor assembles increasing portions of the software results in releases that may be monitored. As each of these is prepared, the functions expected can be compared to the functions actually included.



DISCUSSION

This chart shows the number of units for releases 1 and 2 decreasing while the number of units for release 3 is increasing. Investigation by the SPO determined that the contractor was shifting functionality from the earlier releases to the later ones. Also, new functions resulted in the creation of an additional release, release 2A.

NOTES

1. Number of units should remain relatively constant. However, it is common for the number of units to increase as the program's design matures.
2. A significant increase in the number of units per release might indicate the program is having unanticipated growth in the complexity of the functions to be delivered.
3. A decrease in the number of units per release might indicate the program is off-loading functions from earlier releases to later releases.
4. To preserve the readability of code and improve maintainability, software should be divided into practical partitions of functions and subfunctions. It is commonly suggested that software units should be sized at 50 lines of source code and should not exceed 100 lines.
5. Testing should begin integrating pieces of the software as early as possible. Software builds should be encouraged to correspond to operationally useful capabilities as soon as is practical.

SECTION III

APPLICATION OF THE METRICS

The previous section presented eight basic Software Reporting Metrics. These metrics represent areas in which data should be provided by the contractor to aid the System Program Office (SPO) in monitoring the software development effort. The data required to support each metric is intended to be generic and to address the project status at a high level. These generic data requirements should be regarded as a minimal set. To be more effective, the metrics should be analyzed to see where additional data would provide better insight for a particular program. The data requirements presented should be expanded wherever appropriate, depending on the risk areas, the nature of the software acquisition, and the existing software monitoring metrics being used by the contractor. This section discusses how this analysis can be performed, illustrates some examples of how the metrics may be expanded, and discusses the actual use of the metrics in the SPO.

1. DEFINING THE DATA TO BE COLLECTED

As mentioned in the introductory paragraph in Section II, the metrics collection is accomplished by preparing Data Item Descriptions (DIDs) or Program Management Review (PMR) deliverables that can be placed on contract. This description should be as clear and precise as possible to prevent misinterpretation on the part of the contractor when providing the data, and on the part of the SPO when processing and analyzing the data. A set of data items that match the eight generic metrics has been included in Appendix B. These data items define, for each metric, the particular data that DID(s) or PMR deliverables should require the contractor to provide. They indicate the level of detail needed, but they are not necessarily complete for a specific project and only support reporting at a high level. Definitions are provided in Appendix D for certain data items.

Regardless of whether the metrics are augmented or not, an analysis of the data items and their appropriateness should always be undertaken. The contractor may have his own methodology for monitoring his progress. This should be taken into account to ensure that the metrics collection meets the minimal set, while not causing the contractor to produce duplicate reports to satisfy his own metrics as well as the government's. For example, if he counts SLOC for his own monitoring of Software Size, but defines SLOC differently from the definition initially selected by the SPO, it may be more cost effective and just as useful to report Software Size according to the contractor's definition. The important issue is to ensure that both parties agree on the definition of what is being reported.

The metrics will not replace dialog and periodic reviews, but augmenting the metrics to obtain additional data can help the SPO in identifying areas that require further investigation and dialogue. The following paragraphs discuss how each metric may be expanded or augmented for sample cases. This is presented only as an illustration of the methodology for performing this expansion rather than as guidance on the particular data that should be added.

It may be desirable to expand several of the metrics when multiple CSCIs exist. The level of detail presented in the generic metrics is at the program level. Often more detail is required because the CSCIs vary considerably in expected size and complexity. Collecting the appropriate metrics at this level may provide a clearer picture of the problem areas and actual software development progress. In addition, if CSCIs are used as a basis for subcontracting portions of the job, collecting data at this level would provide insight into subcontractor performance. The data then can be aggregated to the program level. Therefore, whenever multiple CSCIs or subcontractors exist, it may be appropriate to expand the metrics. This may require adding data to each metric to identify the CSCI.

The Software Size metric collects data on lifted, modified, and new SLOC. An appropriate expansion of this metric may be to collect this data based on the source language in cases where software is being delivered in more than one language. This expansion would require adding a field to the software size metric for a source code language identifier.

The Software Personnel metric collects data on years of experience to gauge whether the contractor is staffing the job with the appropriate mix of personnel. In some cases, it may be appropriate to expand the categories for which staffing is reported. For example, if particular skills such as Ada, DBMS or AI are needed to successfully complete the effort, the SPO may want to add a field to indicate the level of experience in each skill area. Another expansion may be to report staffing according to the type of task, such as testing, to ensure that critical tasks are staffed appropriately.

The Software Complexity metric may be expanded to include other complexity measures. For example, Halstead's (Ref. 2) or McCabe's (Ref. 3) complexity measures can be used after CDR in addition to the complexity estimates derived from Table 1.

The Development Progress metric can be expanded to collect data at the CSC level as well as the unit level. A project may wish to be able to track the development progress of the CSCs by adding a field that reports the number of units designed/tested/integrated for each CSC. The project

may only wish to do this for CSCs that are part of a critical CSCI. In addition, an expansion may include tracking the number of units for which Program Design Language (PDL) code has been developed to aid in monitoring the design progress.

The Testing Progress metric can be expanded by tracking the length of time a problem report is open, as well as tracking how many are open. This can help in identifying how quickly problem reports are being addressed. This also would give some indication of the number of complex problem reports, because an older report that has not been closed may have remained open due to its difficulty.

It may be desirable to expand the Computer Resource Utilization metric to track utilization for each resource unit. The rationale for this is that, for example, without separate reporting for each processor in a multiprocessor system, the total amount of CPU time consumed might be an average across processors. This figure may be below the desired spare, but one particular processor may be well above the limit, while another is significantly below. Tracking by resource unit would alleviate this confusion. In addition, both average and worst case utilization estimates could be collected. Another possible expansion for this metric is to collect data on both target and development machines, if the contractor plans to develop the software on a machine that is different from the target machine.

The Software Volatility metric counts the number of SLOCs affected by an ECP, as well as the number of open review discrepancies. It may be desirable to also track the length of time that a review discrepancy has been open. The rationale for this expansion is similar to that given for tracking the length of time that problem reports are open. In addition, the Software Volatility metric could be expanded to track the number of units that are affected by an ECP, as well as the number of SLOC.

The Incremental Release Content metric tracks the number of units planned for each release in order to monitor whether any capabilities are being shifted from an earlier release to a later release. The metric can be expanded to also track the number of CSCs in each release as another gauge of capabilities to be included in any given release.

The above paragraphs represent sample ways in which the metrics can be expanded to address specific areas of concern for a project. Data items have been included in Appendix C to illustrate the data field definitions necessary to insure collection of an expanded set of metrics. The reader is cautioned that these data items are intended to be representative; each program that implements the metrics should be careful to analyze which, if any, of these expansions are appropriate, and identify other expansions as needed.

2. IMPLEMENTATION OF DATA COLLECTION

Once data requirements to support metrics reporting have been defined, the SPO must identify mechanisms for actual delivery of the metrics data from the contractor. Two schools of thought exist on how the contractor should provide the data. One opinion is that the contractor should supply the data in digital format. This could be accomplished by requiring that the data be submitted on a magnetic tape or floppy disk each reporting period. Another mechanism for digital collection could be via electronic data transfer from the contractor's computer system to a computer system available to the SPO. Potentially, most of the data that is required should already exist in digital form in the contractor's configuration, personnel, and other management systems. Delivery of the data in digital format, therefore, should not provide any additional burden for the contractor, and may in fact reduce the effort necessary to provide the data. In addition, the SPO can process the data on-line instead of translating data in hard copy form to digital form. The major advantage of such a collection mechanism is in being able to quickly process and graph the metrics data upon receipt. Assuming the SPO had some computer system that would support these functions (see discussion below), the time for clerical duties regarding making the data available would be reduced, providing more time for actual analysis.

The second opinion is that the contractor should provide the data in the form of written CDRL submissions or as a part of his presentation at Program Management Reviews. The rationale here is that the SPO would have to extract this information and, in the process, would become more conscious of the data and its analysis. This school of thought is prompted by concern that the automated approach will lead to inaccuracies in the data reported, misinterpretations of the data, and will thus detract from the level of attention that should be given to the metrics data analysis.

There are tradeoffs involved in either collection mechanism. The SPO must decide which mechanism is appropriate and the degree of integrity that can be achieved when the data is collected manually versus electronically.

3. AN AUTOMATED CAPABILITY FOR METRICS PROCESSING

Once data definition and data collection mechanisms have been defined, and data is being delivered in some form to the SPO, the SPO must have a capability to store, process, and plot the data. The following paragraphs describe a preliminary prototype effort to develop such a capability.

The architecture selected for the prototype uses a relational database on a centralized processor connected via data communications to a workstation with spreadsheet and graphics capabilities. By utilizing a relational database on a centralized processor, the extensibility of the

system is enhanced. Metrics data for many projects can be stored in a common location enabling access to data across projects. This enables the formulation of a historical database that could be analyzed for "lessons learned." In addition, other data elements can be added to a particular project's database to support common use of the project database among different software acquisition tools. This environment permits data sharing, reduces data replication, and increases data integrity.

The centralized database can be updated via manual data entry, electronic transfer from another system, or via tape. To accomplish these tasks, it is possible to use the workstation as a dumb terminal connected to the central processor. Once the centralized database has been updated, the data can be downloaded to the workstation where it can be graphed and manipulated. Downloading data to the workstation, allows the use of a spreadsheet package to manipulate the data independently of the centralized database. This enables the SPO to alter the data for trade-off analyses without affecting the integrity of the centralized database. The spreadsheet package provides the capabilities to produce graphical displays similar to those contained in the previous section.

The user's view of the system is simplified by command files and spreadsheet macros that reduce the user interaction to a minimum. To use the system, a command file is invoked on the workstation. It executes a communications package that allows the user to log on to the centralized processor. Once logged on, the user has control of a menu driven command process that allows him to enter, update, query, or extract data from the database. The extraction routine not only extracts data from the database, but orchestrates its downloading to the workstation. When the user has completed his centralized processor tasks he can terminate the connection. The workstation command file then executes the spreadsheet package. The user selects the spreadsheet corresponding to the metric that he wishes to process. A number of spreadsheet macros handle the importing of the appropriate data and setting the ranges that are required for the graphic displays.

The initial implementation of the prototype uses a VAX 11/780, running VMS with the Rdb relational DBMS, as the centralized processor. The workstation is a Z-100 running Z-DOS with PC/Intercomm (a communications package) and LOTUS 1-2-3 (a spreadsheet package with macro and graphical display capabilities).

SECTION IV

CONTINUING METRICS EFFORTS

Revision 2 of the Software Reporting Metrics document was prepared in response to comments received from a variety of Air Force, MITRE, and industry personnel. However, many outstanding issues remain to be addressed. This section outlines some of the outstanding issues by first presenting general topics to be addressed, followed by a list of specific comments requiring further research. This is followed by a brief discussion of other activities to be pursued to enhance application of the metrics.

1. GENERAL COMMENTS TO BE ADDRESSED

The following paragraphs present general topics for refining and augmenting the original metrics.

It is recognized that there are a significant number of related efforts being pursued to measure the quality of the software acquisition process and the software product itself. Some examples are the RADC metrics work, the STARS measurement activity, the AFSC indicators, and the IEEE metrics efforts. A comparison of these metrics with the metrics presented in this report should be undertaken to identify how the results of these efforts can be transitioned into practice.

Another activity is to perform case studies of software acquisitions that have used the metrics and use the feedback received to improve the metrics. This activity would also be useful to validate the metrics.

A study should be made to investigate the context of the metrics to identify where they fit in the acquisition process and how they complement and relate to one another. The existing metrics tend to concentrate on the development phase of software acquisition by collecting more information after CDR and before FQR. Front-end and back-end metrics may be needed to address the requirements definition and maintenance phases, respectively, and to effectively monitor the entire software acquisition effort.

The addition of project management metrics and quality metrics also has been suggested. The present metrics are status-oriented and do not necessarily address the quality of management or the quality of software deliverables. They address status as a means of

monitoring progress, but do not consider the quality of the monitoring process nor the software itself. Some effort to define additional metrics that address these areas should be pursued.

Another comment, related to the one above, suggests the addition of a matrix of process metrics versus product metrics. Process metrics examine the activities that encompass the production of the product while product metrics examine the finished product. It would be useful to identify those metrics that address each of these activities.

2. SPECIFIC COMMENTS TO BE ADDRESSED

The following paragraphs are comments on specific metrics that were received but not integrated into this revision. Further investigation is required to integrate these comments.

The subjectivity of the Software Complexity metric was a subject of several comments. Replacing or augmenting the existing complexity measure with one that is less subjective than the measure based on Boehm's work would address these comments. A revised Software Complexity metric would probably draw upon the complexity measuring work of Halstead (Ref. 2), McCabe (Ref. 3), and Albrecht (Ref. 4).

The addition of other data to increase the value of the Development Progress metric was suggested. This data might include CSCI status, CSC status, B-5 status, C-5 status, and a count of the number of patches added to get through testing. Defining how to track the status of these items and how to interpret the data are left as future activities.

Another comment suggested using "threads" as the item to be tracked in the Testing Progress metric. The use of threads would add a functional perspective to the testing. It is felt that a function test is more meaningful than a unit test as a gauge of testing progress.

The Testing Progress metric also could be modified to count the number of SLOC affected by problem reports. This would give an indication of the severity or effort required to resolve the problem. To incorporate this comment will require an understanding of what constitutes "affected SLOC".

ECPs have been identified as insufficiently responsive to act as a leading indicator of software volatility. In addition, measuring the SLOC affected by an ECP adds a degree of uncertainty and subjectivity to the metric. Measures of Issue Reports (also called Investigation Requests) have been proposed as a potential substitute. These reports identify problems in requirements found while establishing the system baseline.

3. FUTURE ACTIVITIES IN APPLICATION OF THE METRICS

Section III of this document presented some guidance on how to apply the metrics to a particular project and the use of automated aids. The following paragraphs briefly discuss areas for expansion in application of metrics on a project.

One area that must be considered is how to integrate the metrics information with other status reports and tools used by the SPO to monitor a software acquisition. These include the Cost Performance Reporting (CPR), cost estimation, and program scheduling information and tools. This information and the tools that are used have a high potential for integration with the metrics to maximize their use. For example, it is possible that more accurate and up-to-date cost and schedule estimates could be derived throughout the acquisition by using the metrics data as an input. In addition, the Cost Performance Report contains an extensive amount of information for the SPO to use in monitoring the acquisition. Efforts to identify areas for correlation and integration of this data with the metrics will be pursued as a future activity.

Another planned area for investigation is to expand the existing metrics workstation prototype to improve its features and add additional functions. In addition, integration with other SPO tools is to be investigated and implemented in future prototypes.

REFERENCES

For further reading:

1. Boehm, Barry W. Software Engineering Economics;
Englewood Cliffs, New Jersey,
Prentice-Hall, 1981.
2. Halstead, M.H. Elements of Software Science;
Elsiver, New York, pp. 274-279, 1977.
3. McCabe, T.J. "A Complexity Measure,"
IEEE Transactions on Software
Engineering, Vol. SE-2, No.4,
pp. 308-320, December 1976
4. Albrecht, A.J. and "Software Function, Source Lines of
Gaffney, J.E., Jr. Code, and Development Effort Prediction:
A Software Science Validation," IEEE
Transactions on Software Engineering,
Vol. SE-9, No.6, pp. 639-648, November
1983.
5. Pressman, Roger S. Software Engineering : A Practitioner's
Approach, New York, McGraw-Hill, 1982.

This book provides an introduction to the methodology of acquiring software. Most of the cost estimation information in chapter four is more usefully covered in Boehm's book; but the remainder of this book provides a useful perspective on the process of acquiring or developing software.

APPENDIX A

GENERAL INFORMATION REGARDING SOFTWARE DEVELOPMENT

The metrics described in the Section II were accompanied by some notes that relate to each metric. This section contains additional information that is applicable to many software acquisitions. It has been obtained from personnel involved in previous software acquisitions and from Ref. 1 and Ref. 4. This information is intended to provide general, rather than hard and fast, guidance. Exceptions to these guidelines may be appropriate for specific programs; but whenever exceptions are made it is important to have a detailed understanding of the justifications. Information applicable to the overall acquisition will be presented first. Some additional information is presented in a sequence that corresponds to the stages in the software development cycle.

1. GENERAL

Wide variations in schedule/effort profiles have been followed by different projects. The following general information should not be misused through overly specific interpretation. It is offered here on the premise that some guidance is better than none. For more data see Ref. 1. Contributors to wide variations in schedule/effort performance are the availability and use of modern software development tools such as Program Design Languages (PDLs), automated debugging facilities, data base design tools, etc.

A. Schedule for Software Development

The number of calendar months (M) from the beginning of software design to the end of Formal Qualification Review (FQR) is related to the number of software development Staff Months (SM) necessary to complete the software. (Complete software refers to software that has been designed, coded, tested, integrated and documented.) The following formulas have been empirically derived from the COCOMO data base. The formulas describe curves for three project percentile values. For example, only 10 percent of the projects in the COCOMO database were able to preserve a schedule equal to or shorter than that schedule predicted by $M = 2 * \sqrt{SM}$.

	Percentile
$M = 2 * \sqrt[3]{SM}$	10%
$M = 2.5 * \sqrt[3]{SM}$	50%
$M = 3 * \sqrt[3]{SM}$	90%

B. Lines of Code per Staff Month

Typical SLOC/SM values are:

150 for easy code
 70 for medium complexity code
 30 for complex code

Note: The Staff Months (SM) refer to the number of software staff months for design, code, test, integration, and documentation expended from contract award through FQR.

C. Contractor's Resource Availability

The contractor should have access to the necessary computer resources for software development. Among the resources required are development and target systems, system software, management software, application development tools, etc.

D. Allocation of Staff and Schedule for Software Development

These are typical values experienced on embedded software programs similar to the C³I systems developed at ESD. Wide variations from these values have been observed but lessons learned indicate that increases in design effort lead to reduced integration and test effort and to higher quality products.

	<u>Staff Months</u>	<u>Schedule</u>
Design	45%	50%
Code and Unit Test	20%	15%
Integration and Test	35%	35%

2. PHASE SPECIFIC

A. Pre-Award

1. Resource Availability:

Software development resources should be evaluated in a manner similar to that for hardware manufacturing facilities:

- (a) Large software development projects should use modern methodologies with automated support tools for software development.
- (b) Sources for experienced software engineering support must be identified and available during the designated software development time period.

2. Requirements Definition:

System requirements must not be confused with design or implementation details or vice versa.

3. Contract Preparation:

The Instructions for Proposal Preparation (IFPP), Statement of Work (SOW) and Contract Data Requirements List (CDRL) should be structured to require the contractor to deliver the data for the reporting metrics.

B. Source Selection

1. Difficult and risky functions should receive early and adequate attention. Workable solutions should be demonstrated before start of Full Scale Engineering Development (FSED).

2. The software development schedule needs to accommodate the practicalities of the process. It should not be "success oriented." Schedule compression adds greatly to cost and does not generally produce an earlier product.

3. The difficulty of a project increases as more organizations, government as well as contractor, participate. The Government should avoid becoming an interface between two or more contractors.

4. The Government should assure that any software development by subcontractors is closely managed by the prime contractor.

5. Large development projects should have functional capabilities implemented in phased releases. If schedule problems develop, useful capabilities can be provided before the development program is completed.

C. Preliminary and Critical Design Reviews

1. Development should not be allowed to proceed faster than the achieved ground work can support. The design should not advance beyond requirements definition, coding beyond the design, testing beyond the stability of the product, etc.

2. Monthly or bimonthly Software Technical Interchange Meetings (TIMs) should supplement the formal reviews. Programs that are large or on tight schedules should have in-plant Government technical representatives so that more frequent contact is maintained between the developers and the acquisition agency.

D. Test and Integration

1. Software testing should be done by a separate and independent organization from that which developed the software.

2. Formal software acceptance testing should allow for time and access to the equipment in which the Government can exercise the system.

3. Integration test planning should be completed early enough to: 1) ensure test issues can influence design, and 2) allow long-lead-time test facilities to be acquired. For most programs, test planning should have started before PDR.

3. REMINDER

Metrics should not be substituted for dialog. When a metric indicates a potential problem, that should be a motivator for dialog.

APPENDIX B

GENERIC METRIC DATA ITEMS

The following list of data items reflects the software reporting metrics as presented in this report. They are a minimal set and may require enhancement for a particular software development effort. If the data items are to be obtained via Data Item Descriptions (DIDs), they should be added to existing DIDs where possible, such as the Software Development Plan DID, to ensure that the same data is not requested twice.

For each metric, the data required for the Current Reporting Period and for the Forecasted Reporting Periods is listed.

Appendix D, Data Definitions, clarifies potentially ambiguous terms.

I. SOFTWARE SIZE

A. Current Reporting Period

1. Current Reporting Period
2. The estimated total of SLOC newly developed for this system
3. The estimated total of SLOC modified for this system
4. The estimated total of SLOC lifted for this system

B. Forecasted Reporting Periods

Not applicable

II. SOFTWARE PERSONNEL

A. Current Reporting Period

1. Current Reporting Period
2. The actual total number of Staff during the period
3. The actual number of experienced Staff during the current period
4. The actual number of Staff leaving the software development effort during the current period
5. The actual number of Staff added to the software development effort during the current period

B. Forecasted Reporting Periods

1. Current Reporting Period
2. Forecasted Reporting Period
3. The estimated total number of Staff during the forecasted period
4. The estimated number of experienced Staff during the forecasted period

III. SOFTWARE COMPLEXITY

A. Current Reporting Period

1. Current Reporting Period
2. Weighted Complexity Rating based on Barry W. Boehm's Table of Complexity Ratings and the relative size of each CSCI

B. Forecasted Reporting Periods

Not applicable.

IV. DEVELOPMENT PROGRESS

A. Current Reporting Period

1. Current Reporting Period
2. The actual number of units designed during the current reporting period
3. The actual number of units tested during the current reporting period
4. The actual number of units integrated during the current reporting period

B. Forecasted Reporting Periods

1. Current Reporting Period
2. Forecasted Reporting Period
3. The estimated number of units to be designed during the forecasted reporting period
4. The estimated number of units to be tested during the forecasted reporting period
5. The estimated number of units to be integrated during the forecasted reporting period

V. TESTING PROGRESS

A. Current Reporting Period

1. Current Reporting Period
2. The actual number of tests passed during the current period
3. The actual number of new testing problem reports occurring during the current period
4. The actual number of testing problem reports unresolved during the current period
5. The actual number of testing problem reports resolved during the current period

B. Forecasted Reporting Periods

1. Current Reporting Period
2. Forecasted Reporting Period
3. The estimated number of tests to be passed during the forecasted period

VI. COMPUTER RESOURCE USAGE

A. Current Reporting Period

1. Current Reporting Period
2. Computer Resource Identifier
3. The estimated percentage of CPU usage for the system
4. The estimated percentage of Memory usage for the system
5. The estimated percentage of I/O usage for the system

B. Forecasted Reporting Periods

Not applicable

VII. SOFTWARE VOLATILITY

A. Current Reporting Period

1. Current Reporting Period
2. The estimated total number of SLOC affected by ECPs during the current period
3. The number of newly opened review discrepancies occurring during the current period
4. The number of review discrepancies not resolved during the current period

B. Forecasted Reporting Periods

Not applicable.

VIII. INCREMENTAL RELEASE CONTENT

A. Current Reporting Period

1. Current Reporting Period
2. Release Identifier
3. The number of units in the release

B. Forecasted Reporting Periods

Not applicable

APPENDIX C

ENHANCED METRIC DATA ITEMS

The following list of data items will assist those persons writing or modifying a Data Item Description (DID) or calling for Program Management Review (PMR) deliverables for the collection of the metrics presented in this report for a particular software development effort.

These data items illustrate the expansion of the Generic Metric Data Items in Appendix B. The customization is necessary when adapting the metrics to a particular software development effort. For each metric, the data required for the Current Reporting Period and for the Forecasted Reporting Periods is listed. Appendix D, Data Definitions, clarifies potentially ambiguous terms.

I. SOFTWARE SIZE

For each CSCI:

A. Current Reporting Period

1. Current Reporting Period
2. CSCI Identifier
3. Source Code Language
4. The estimated total of SLOC newly developed
5. The estimated total of SLOC modified
6. The estimated total of SLOC lifted

B. Forecasted Reporting Periods

Not applicable

II. SOFTWARE PERSONNEL

For each CSCI:

A. Current Reporting Period

1. Current Reporting Period
2. CSCI Identifier
3. For each experience category, the actual number of staff in the experience category during the current period

4. The actual number of Staff leaving the software development effort during the current period
5. The actual number of Staff added to the software development effort during the current period

B. Forecasted Reporting Periods

1. Current Reporting Period
2. Forecasted Reporting Period
3. CSCI Identifier
4. For each experience category, the forecasted number of staff in the experience category during the forecasted period

III. SOFTWARE COMPLEXITY

For each CSCI:

A. Current Reporting Period

1. Current Reporting Period
2. CSCI Identifier
3. Weighted Complexity Rating based on Barry W. Boehm's Table of Complexity Ratings and the relative size of each CSCI
4. Thomas McCabe's cyclomatic complexity rating

B. Forecasted Reporting Periods

Not applicable.

IV. DEVELOPMENT PROGRESS

For each CSC:

A. Current Reporting Period

1. Current Reporting Period
2. CSCI Identifier
3. CSC Identifier
4. The actual number of units designed during the current reporting period
5. The actual number of units tested during the current reporting period
6. The actual number of units integrated during the current reporting period

B. Forecasted Reporting Periods

1. Current Reporting Period
2. Forecasted Reporting Period
3. CSCI Identifier
4. CSC Identifier
5. The estimated number of units to be designed during the forecasted reporting period
6. The estimated number of units to be tested during the forecasted reporting period
7. The estimated number of units to be integrated during the forecasted reporting period

V. TESTING PROGRESS

For each CSCI:

A. Current Reporting Period

1. Current Reporting Period
2. CSCI Identifier
3. The actual number of tests passed during the current period
4. The actual number of new testing problem reports occurring during the current period
5. The actual number of testing problem reports resolved during the current period
6. The actual number of testing problem reports unresolved during the current period
7. The actual number of testing problem reports unresolved during the current period that are older than two but less than three reporting periods
8. The actual number of testing problem reports unresolved during the current period that are older than three reporting periods

B. Forecasted Reporting Periods

1. Current Reporting Period
2. Forecasted Reporting Period
3. CSCI Identifier
4. The estimated number of tests to be passed during the forecasted period

VI. COMPUTER RESOURCE USAGE

A. Current Reporting Period

1. Current Reporting Period
2. Category (Development or Target)
3. Computer Resource Identifier
4. The estimated percentage of average CPU usage for the system
5. The estimated percentage of average Memory usage for the system
6. The estimated percentage of average I/O usage for the system
7. The estimated percentage of worst case CPU usage for the system
8. The estimated percentage of worst case Memory usage for the system
9. The estimated percentage of worst case I/O usage for the system

B. Forecasted Reporting Periods

Not applicable

VII. SOFTWARE VOLATILITY

For each CSCI:

A. Current Reporting Period

1. Current Reporting Period
2. CSCI Identifier
3. The estimated total number of SLOC affected by ECPs during the current period
4. The estimated total number of units affected by ECPs during the current period
5. The number of newly opened review discrepancies occurring during the current period
6. The number of review discrepancies not resolved during the current period
7. The number of review discrepancies not resolved during the current period that are older than two but less than three reporting periods
8. The number of review discrepancies not resolved during the current period that are older than three reporting periods

B. Forecasted Reporting Periods

Not applicable

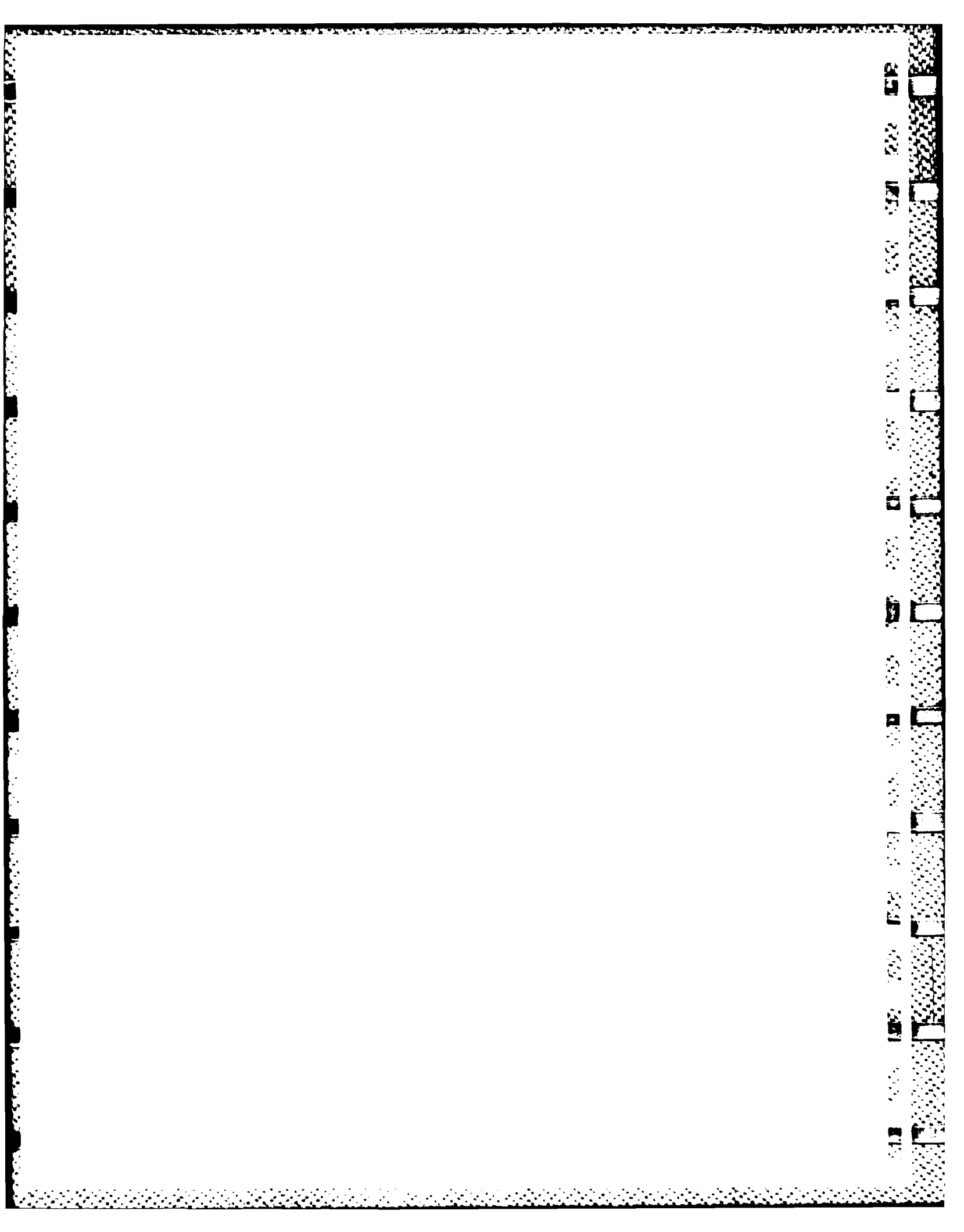
VIII. INCREMENTAL RELEASE CONTENT

A. Current Reporting Period

1. Current Reporting Period
2. Release Identifier
3. The number of units in the release

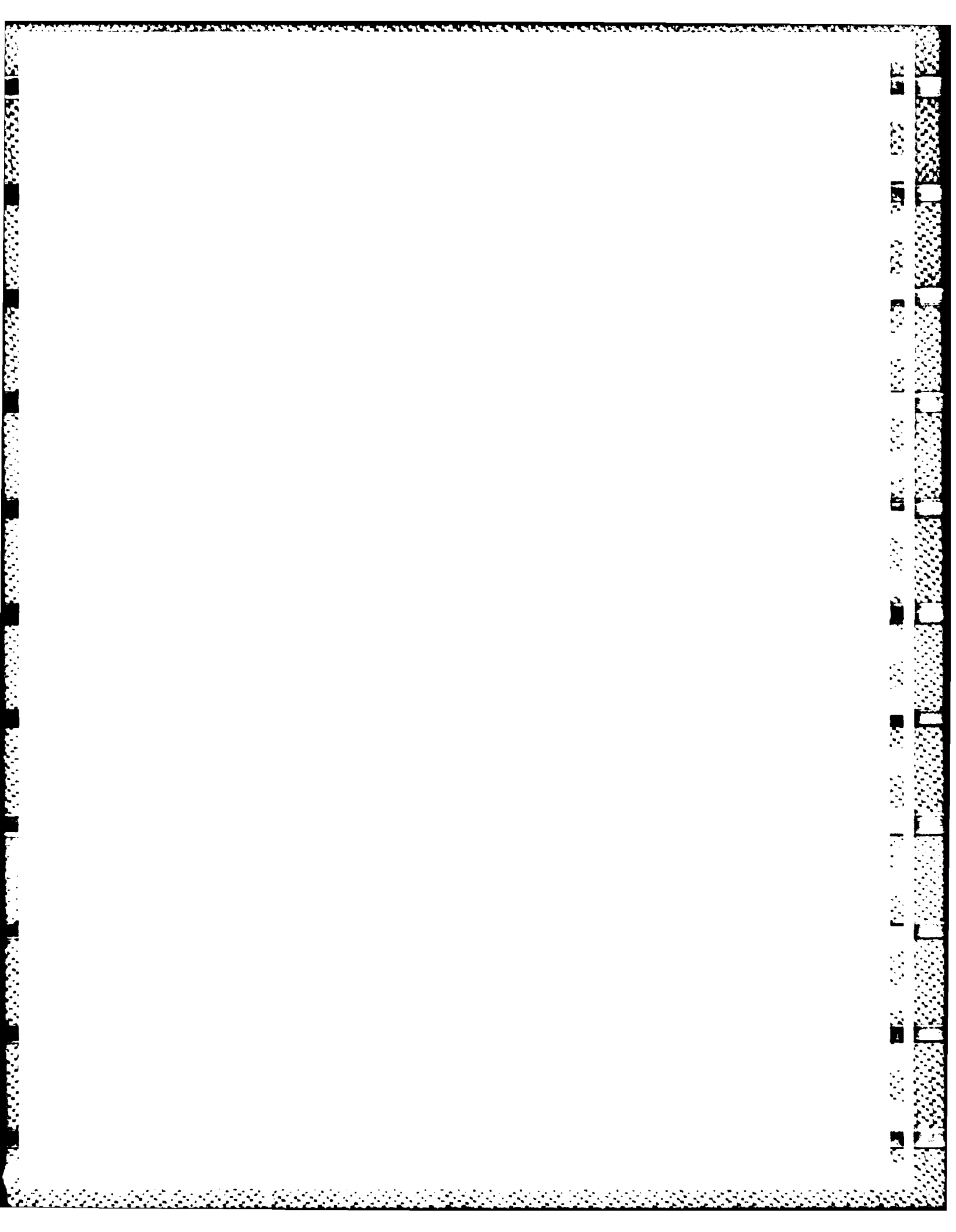
B. Forecasted Reporting Periods

Not applicable



APPENDIX D
DATA DEFINITIONS

SLOC:	Source Lines of Code - all executable statements Note: a tighter definition should be developed depending on the source code language. Also, SLOC could be broken down into categories: Comments, Data Declarations, and Executable Statements.
Lifted SLOC:	SLOC 'copied' from another source, then reused without change.
Modified SLOC:	SLOC 'copied' from another source, modified, then used.
New SLOC:	SLOC newly developed, that is, not copied from another source.
Experience Category:	These are categories that define the different degrees of experience and capabilities of the software development staff.
Reporting Period:	A symbol for denoting the intervals for which the software reporting metrics are collected. For example, if the reporting period is monthly, the reporting period could be the number of the month after contract award.
Source Code Language:	The language used during software development, for example, PL/I, Ada, Assembler, and so on.
Staff:	All those directly involved in the software development effort including Project Manager, Project Leader, Programmer, Quality Assurance Staff, and Configuration Management Staff. Also, a person or the fractional amount of a person's time devoted to the software development effort; for example, a staff member who is devoting only half time to this program would be counted as (.5).
Target Computer Resource:	The computer resource to be used in the delivered operational system.



END

FILMED

3-86

DTIC