

DTIC FILE COPY

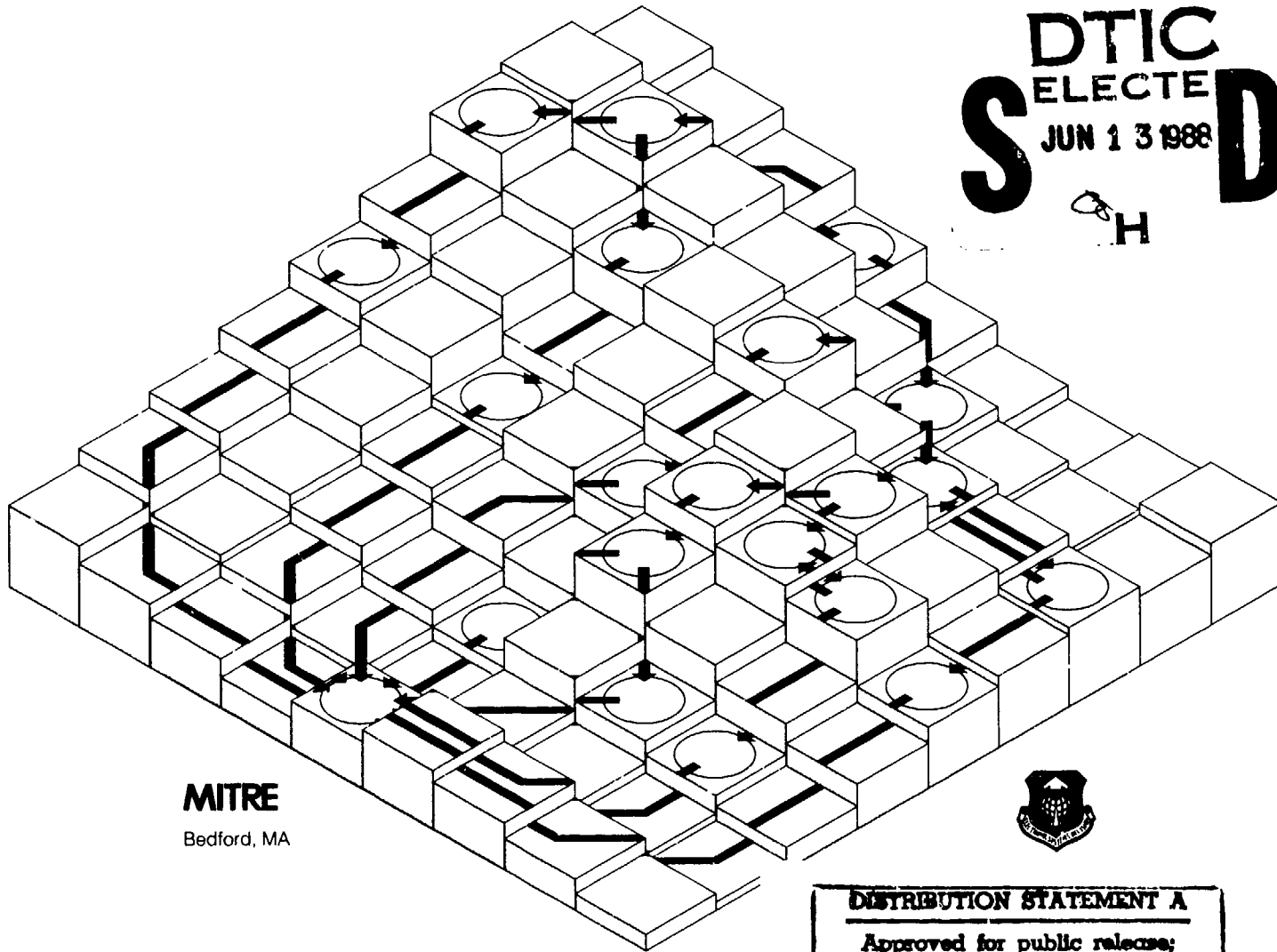
3

AD-A196 916

Software Management Metrics

Herman P. Schultz • May 1988

DTIC
ELECTE
JUN 13 1988
S H D



MITRE
Bedford, MA

**DISTRIBUTION STATEMENT A**

Approved for public release;
Distribution Unlimited

Prepared for Deputy Commander for Product Assurance and Acquisition Logistics
Electronic Systems Division, AFSC, United States Air Force, Hanscom Air Force Base, Massachusetts
Approved for public release, distribution unlimited.

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.

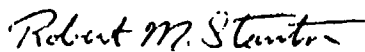
REVIEW AND APPROVAL

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



GARY M. SHEINFELD
Chief
Software Acquisition Management Division
Systems Engineering

FOR THE COMMANDER



ROBERT M. STANTON
Director
Systems Engineering
Deputy for Product Assurance
and Acquisition Logistics

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

ADA196916

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) 88-1 ESD-TR-88-001			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 Commander (continued)		8b. OFFICE SYMBOL (If applicable) ESD/PLE		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 NUMBERS			
		PROGRAM ELEMENT NO.	PROJECT NO. 565B	TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) Software Management Metrics					
12. PERSONAL AUTHOR(S) Schultz, Herman P.					
13a. TYPE OF REPORT Final		13b. TIME COVERED FROM TO		14. DATE OF REPORT (Year, Month, Day) 1988 May	
15. PAGE COUNT 60					
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) Management Indicators, Metrics, Process Metrics, (continued)		
FIELD	GROUP	SUB-GROUP			
19. ABSTRACT (Continue on reverse if necessary and identify by block number) The metrics presented in this document are used to monitor the progress of a software development effort. They provide visibility into developing trends and thereby can be used to forecast potential problems. These metrics are based on Government and industry experiences with a previous set of metrics known as "Software Reporting Metrics" and on an evaluation and analysis of their use. The term "reporting" has been changed to "management" to more precisely describe the metrics' application. This new report includes: a total of 10 metrics instead of 8; 4 new metrics (design progress, design complexity, schedule progress, and requirements volatility); reporting and analysis recommendations; structured descriptions for each metric that include tailoring, interpretation, and behavior discussions; sample Data Item Description (DID) backup sheets for requiring metrics; notes on applications to Ada; and charting and presentation recommendations. <i>Keywords:</i> *Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF RESPONSIBLE INDIVIDUAL Pamela J. Cunha			22b. TELEPHONE (Include Area Code) (617) 271-2844		22c. OFFICE SYMBOL Mail Stop D135

UNCLASSIFIED

UNCLASSIFIED

8a. for Product Assurance and Acquisition Logistics

18. → Project Management,
Reporting Metrics,
Software Acquisition,
Software Metrics
Status Monitoring,

Systems management, Systems
Engineering. (SMA) ←

UNCLASSIFIED

ACKNOWLEDGMENTS

This report was prepared by The MITRE Corporation. The work was sponsored by the Systems Engineering Division (PLE), Deputy for Product Assurance and Acquisition Logistics of the Electronic Systems Division (ESD) of the Air Force Systems Command, United States Air Force, Hanscom AFB, Bedford, MA 01731. Funding for this report was provided by Project 5650, Contract No. F19628-86-C-0001, ESD/MITRE Software Center Acquisition 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, regula-

tions, and standards for software acquisitions; 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* [1] and from T. J. McCabe's *Structured Testing* and *A Complexity Measure* [2, 4]. This document has also been revised for compatibility with the new Defense System Software Development Standard, DOD-STD-2167A. The original set of metrics that were the basis for this report were authored by T. F. Saunders of MITRE.



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

TABLE OF CONTENTS

Section	Page
1 Introduction	1
2 Metrics Coverage, Reporting, and Analysis	3
3 The Metrics	11
Software Size Metric	12
Software Personnel Metric	14
Software Volatility Metric	16
Computer Resource Utilization Metric	18
Design Complexity Metric	20
Schedule Progress Metric	22
Design Progress Metric	24
CSU Development Progress Metric	26
Testing Progress Metric	29
Incremental Release Content Metric	32
List of References	35
Bibliography	36
Appendix A General Software Acquisition Information	37
Appendix B Sample DID Backup Sheet	39
Appendix C Enhanced DID Backup Sheet	43
Appendix D Glossary	47
Appendix E Design Complexity Definitions	51

LIST OF ILLUSTRATIONS

Figure		Page
1	Metrics Coverage	3
2	Metrics Correlation Matrix	4
3	Unit Development Progress and Software Personnel	5
4a	CSU Development Progress	6
4b	CSU Development Extrapolation	7
4c	CSU Development Completion	7
5a	Code and Test Progress	8
5b	Test Extrapolation	8
5c	Test Progress with Replan and Actuals	9
6	Number of Integration Tests	9
7	Software Size	13
8	Software Personnel	15
9a	Software Volatility/Software Requirements Changes	17
9b	Software Volatility/SAIs	17
10	Computer Resource Utilization	19
11	Design Complexity	21
12	Schedule Progress	23
13	Design Progress	25
14	CSU Development Progress	27
15a	Testing Progress/Test Status	30
15b	Testing Progress/Software Problem Reports	31
16	Incremental Release Content	33

SECTION 1

Introduction

The Software Management Metrics presented in this document provide a top-level management overview of software development status. These metrics are based on Government and industry experiences with a previous set of metrics known as "Software Reporting Metrics" and on an evaluation and analysis of their use. The metrics can provide early indications of potential software development problems, and can call attention to and stimulate discussion leading to early resolution of those problems. There are 10 metrics — 5 that measure software development progress, and 5 that affect software development progress. The metrics are reported by the contractor at each Program Management Review (PMR).

The successful use of the metrics depends on the program manager's enforcement of a serious technical review of the data collected for each metric. The metrics graphs are a tool for escalating the discussion of important progress and status indicators to both Government and contractor senior managers.

The metrics graphs show developing trends that may indicate future problems and, when analyzed as a set, can highlight inconsisten-

cies that might otherwise be overlooked. Through senior management reviews and related communications, the significance of the identified trends can be determined and appropriate action taken.

Approximately three years of experience in the use and analysis of metrics has resulted in this report, which includes actual examples as well as comments on the behavior and interpretation of each metric.

This document describes the Software Management Metrics and provides information relating experience on previous software acquisitions. Section 2 discusses metrics coverage, reporting, and analysis. Section 3 describes the Software Management Metrics; each description includes a statement of purpose, typical behavior patterns, data inputs, tailoring ideas, example plots, and interpretation notes. Appendices A through E provide general information regarding software development, sample Data Item Description (LID) backup sheets for collecting metrics, data definitions, and design complexity definitions. This document reflects the new DOD-STD-2167A and its associated DIDs.

SECTION 2

Metrics Coverage, Reporting, and Analysis

The Software Management Metrics described in this document are intended to assure that (1) there are metrics to cover all phases of software development; (2) the metrics are delivered in a manner that assures management visibility into important issues; and (3), and most importantly, the metrics reported are effectively analyzed to identify potential software development problems. This section describes the metrics' coverage and provides recommendations regarding their reporting and analysis.

Coverage

The metrics cover all phases of software development. Figure 1 relates the life of each metric to development milestones. As can be seen, all development phases are covered more than once. This multiple coverage provides not only better visibility into each development phase, but also an opportunity to verify the correctness of reported data through consistency checks. The metrics address two aspects of development: progress and planning. The five progress metrics

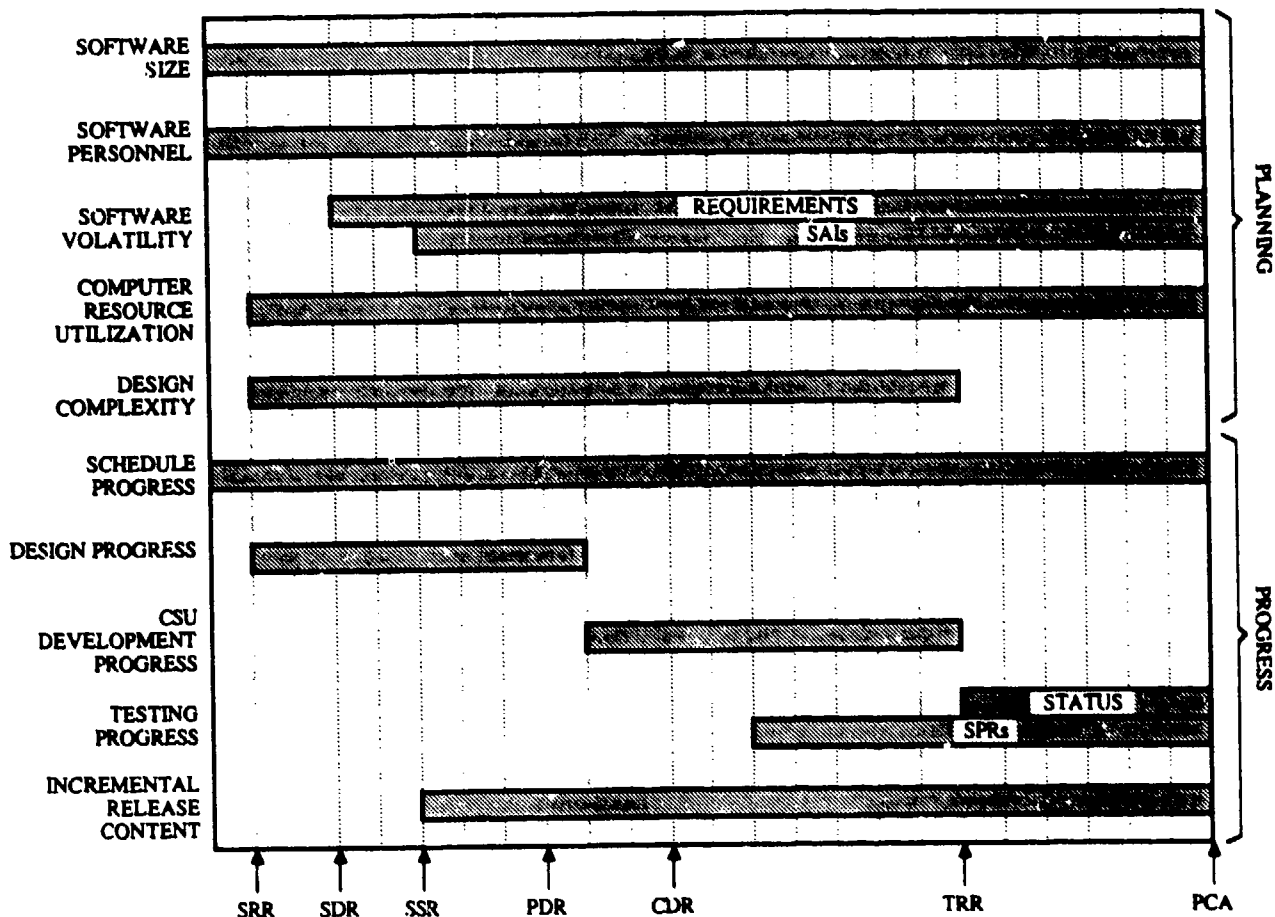


Figure 1. Metrics Coverage

clearly indicate deviations between the planned and actual status of software development. The five planning metrics strongly influence software development progress.

Reporting

The metrics should be presented at each PMR where they can provide management with visibility into potential cost and schedule impact problems. In order to focus the PMR discussions, a pre-PMR metrics screening is recommended that should be accomplished in two steps:

- The contractor delivers the metrics to the Government at least 1 week prior to the PMR. At that time, they are discussed in a Technical Interchange Meeting (TIM) or by telephone conference between the

Government's and contractor's technical staff to identify potential problem areas for discussion at the PMR.

- The results of this TIM are discussed within the System Program Office (SPO) at a meeting with the SPO director. The purpose of this meeting is to separate issues to be discussed at the PMR from those to be discussed at TIMs. The recommendations are conveyed to the contractor, who then tailors his PMR presentation accordingly and schedules any necessary TIMs.

Analysis

The metrics provide a mechanism for evaluating the credibility of project plans and for identifying trends — not only in deviations

IMPACT

CHANGES IN

METRIC	SLOC	PERS	REQT CHGs	SAIs	CRU	DSGN CPLX	SCHD PROG	DSGN PROG	DEV PROG	TEST STAT	SPRs	IRC
SLOC		X			X		X		X	X	X	X
PERS							X	X	X	X		
REQT CHGs	X				X	X	X	X	X	X		X
SAIs		X					X	X	X	S		
CRU				X			X					
DSGN CPLX		X					X	X	X	X	X	X
SCHD PROG												
DSGN PROG							X		S	S		
DEV PROG							X			S		
TEST STAT							X					S
SPRs		X					X		X	X		
IRC							X					

S = SCHEDULE IMPACT

Figure 2. Metrics Correlation Matrix

between planned and actual values, but also in projections that can be made from actual values. Therefore, program managers, in addition to their staffs, must understand if not actively participate in the metrics' analysis in order to draw conclusions that will influence management decisions. Two analysis techniques that have proven to be effective are correlation and extrapolation.

Correlation

It has been shown in figure 1 that during any phase of development, several metrics are being reported. There is also a strong relationship between the reported metrics; that is, changes in one metric should cause changes in others. The matrix in figure 2 indicates the more common relationships. Changes in the metrics listed on the left should have an impact on those whose columns are marked with an X. The analyst should look for inconsistencies within the related group of metrics. For example, if the Development Progress metric shows that

actual development is behind schedule and the Personnel metric shows a reduction in planned staffing, then these metrics are not consistent and should be discussed with the contractor. Figure 3 is an example of just such an occurrence on an actual project. In this case, the contractor acknowledged the discrepancy and revised the staffing plan the following month. The correlations shown in the matrix are not meant to be all-inclusive. Specific projects may have other important relationships.

Extrapolation

One means of analysis that can provide an early indication of potential problems is extrapolation. Trends in actual data can be projected to evaluate their potential impact on schedules. This applies to progress data for development items such as Computer Software Unit (CSU) design, code and test, and integration, as well as to quantitative trends in software size, errors, and requirements changes.

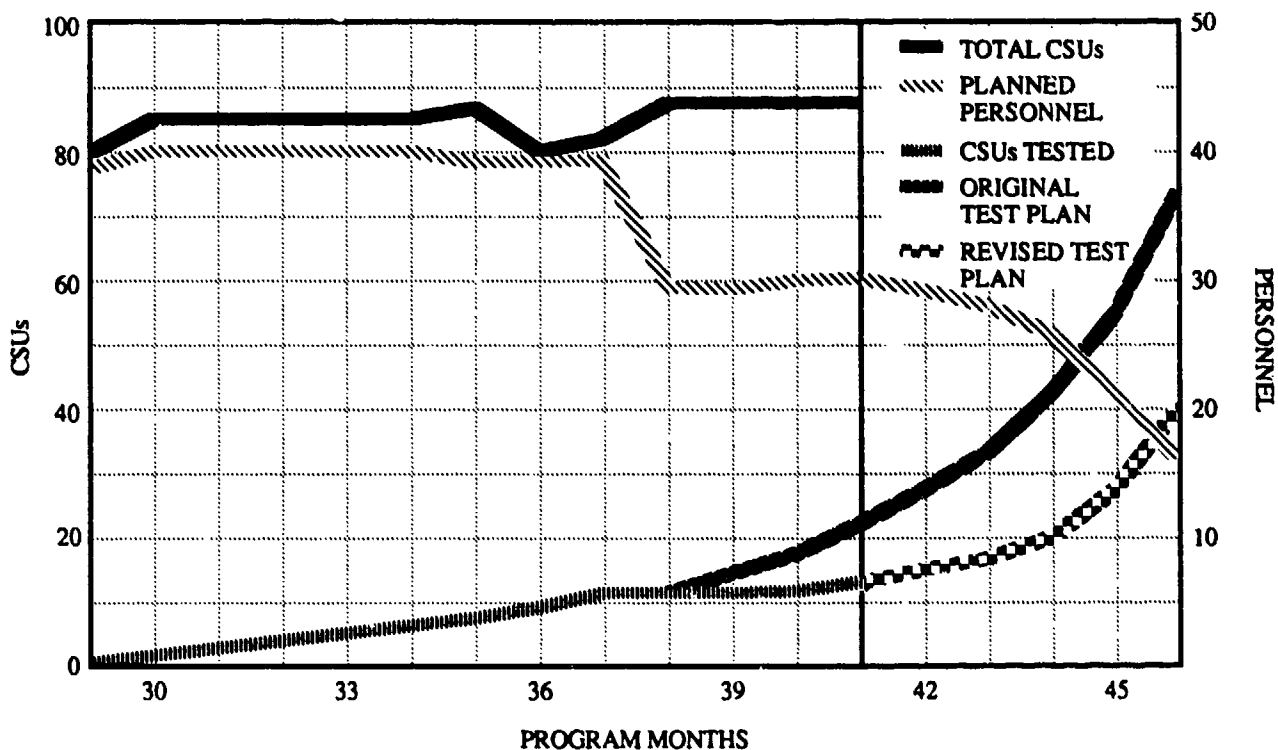


Figure 3. Unit Development Progress and Software Personnel

To show how useful and accurate extrapolation can be, two examples are presented from actual project graphs. These were selected because they contain enough data to show a trend, and in both cases the trend is distinctly different from the plan. Figure 4a is a graph taken from a PMR. Notice that actual progress lags behind the plan. Figure 4b shows an extrapolation based on actual progress to date. This extrapolation predicted the development to be completed around week 55, about 20 weeks behind schedule. In figure 4c, the actual data for the balance of the development was obtained and plotted. The last CSU was actually developed during week 55.

Another example is shown in figures 5a through c. In this example, 4 months elapsed between figures 5b and 5c. The extrapolation was performed with only the data shown in 5a. Actual data for the intervening 4 months was then obtained and, as can be seen, it closely follows the extrapolated schedule. Data for the following 4 months later became available and was added to the graph. Notice how the actual data continues to follow the extrapolated schedule in spite of a replan at month 42.

Figure 6 again illustrates the unvarying nature of an established trend. It shows that an extrapolation made as early as April or May would have been quite accurate.

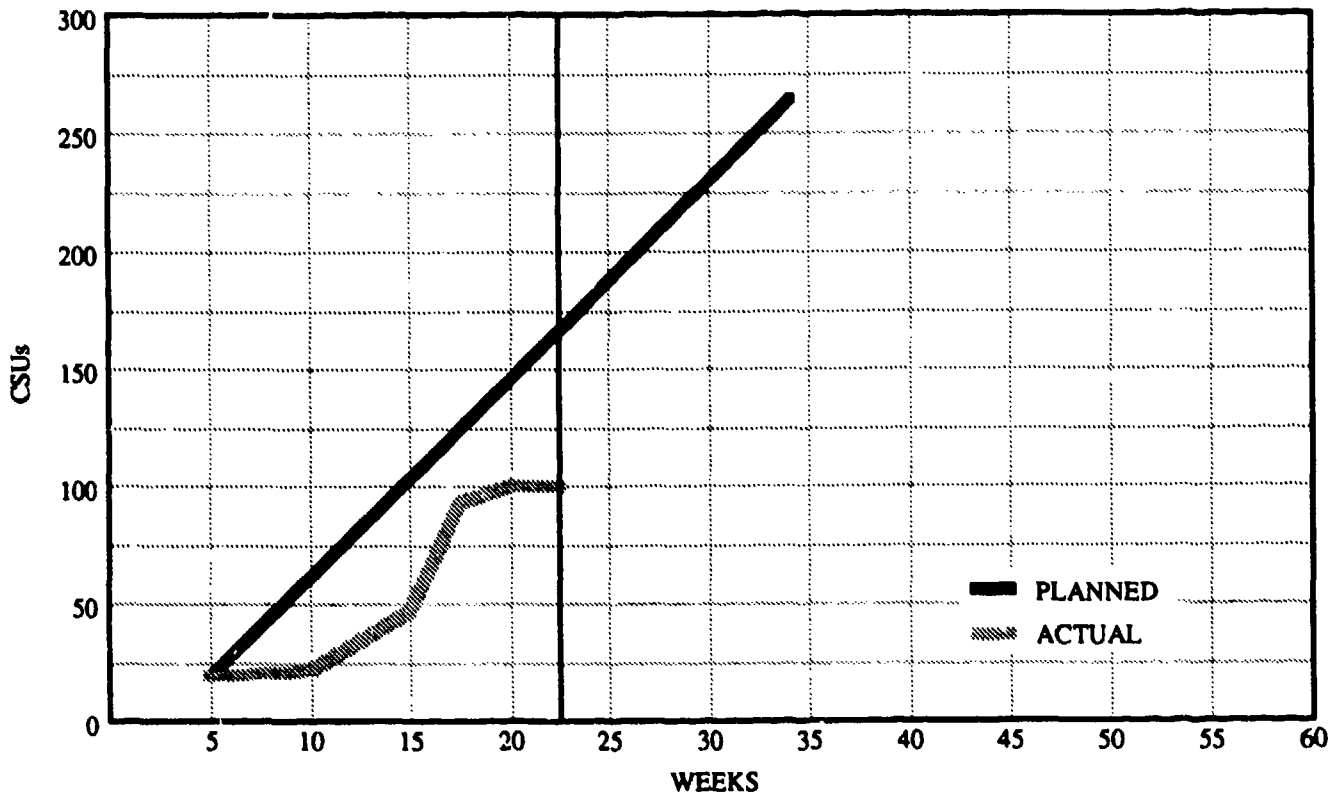


Figure 4a. CSU Development Progress

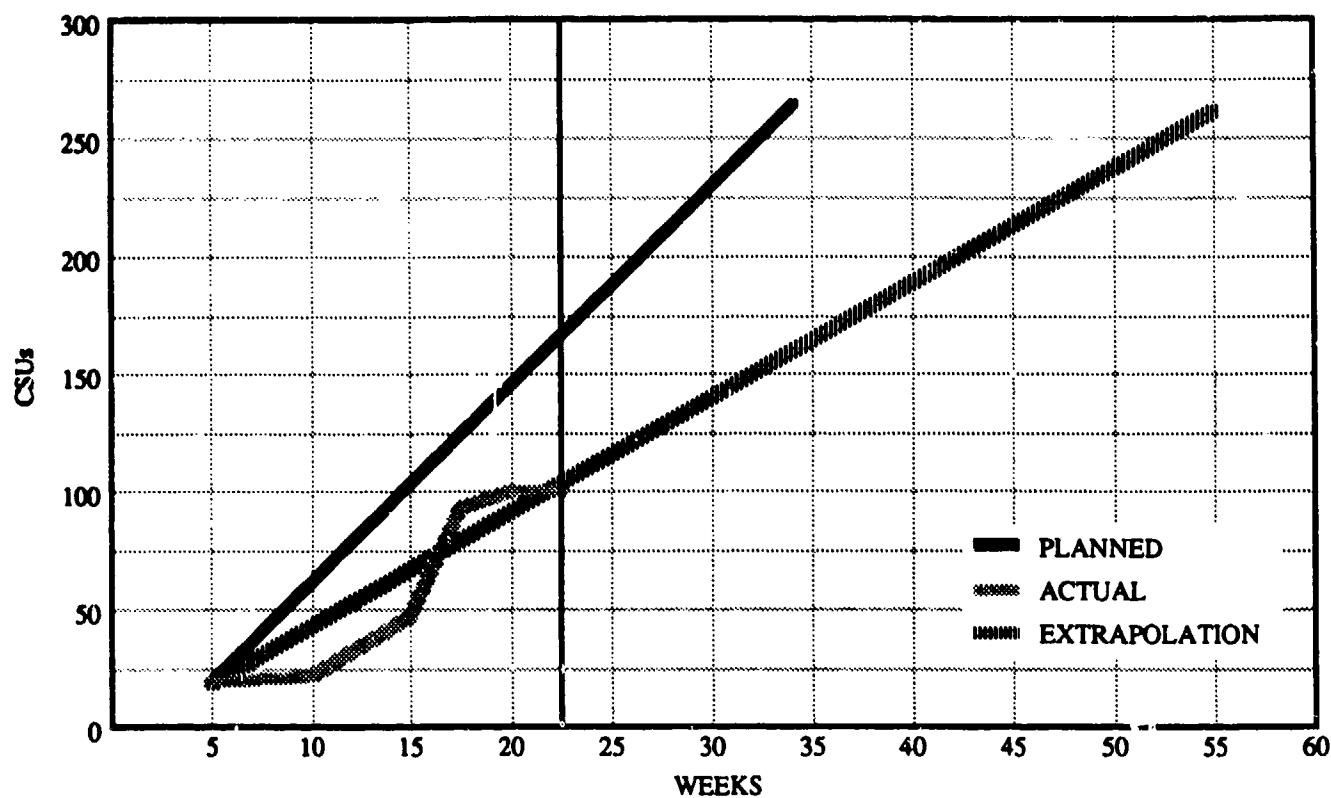


Figure 4b. CSU Development Extrapolation

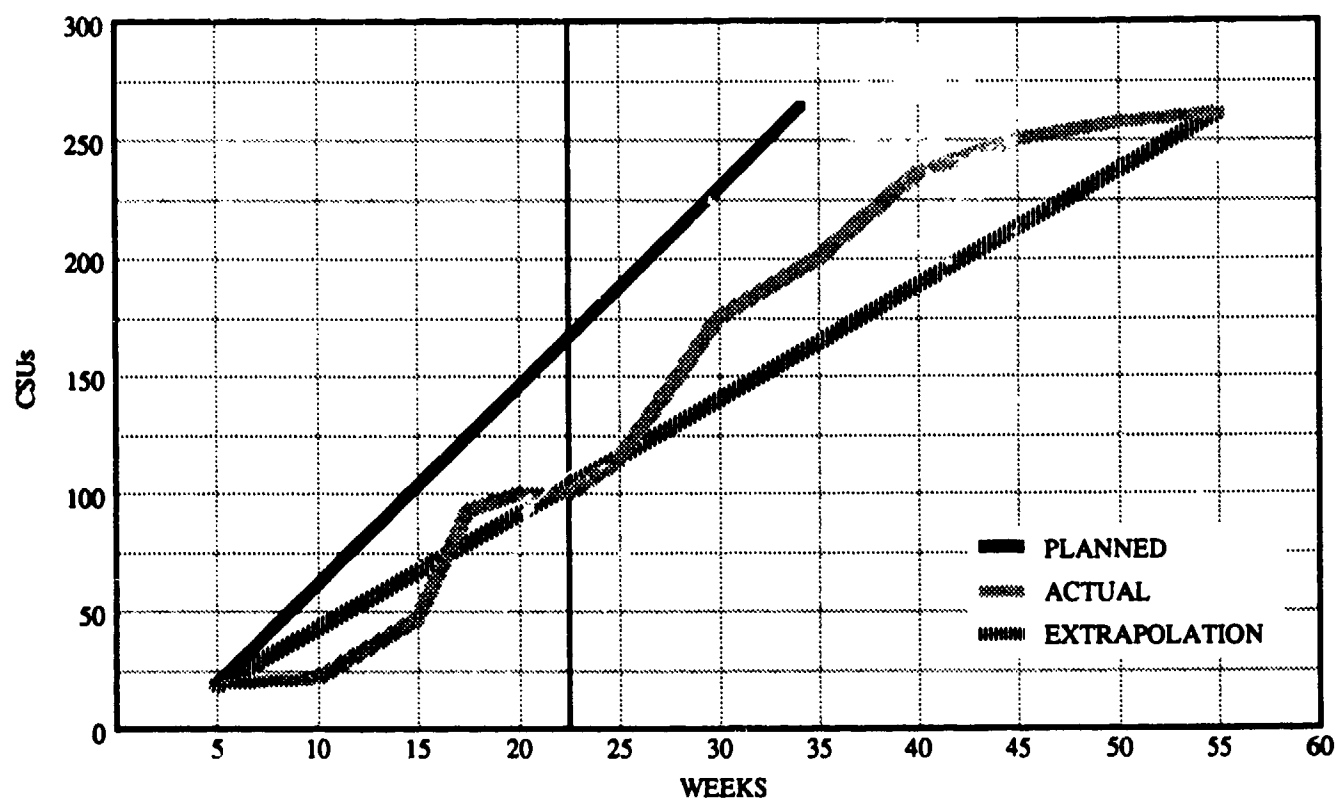


Figure 4c. CSU Development Completion

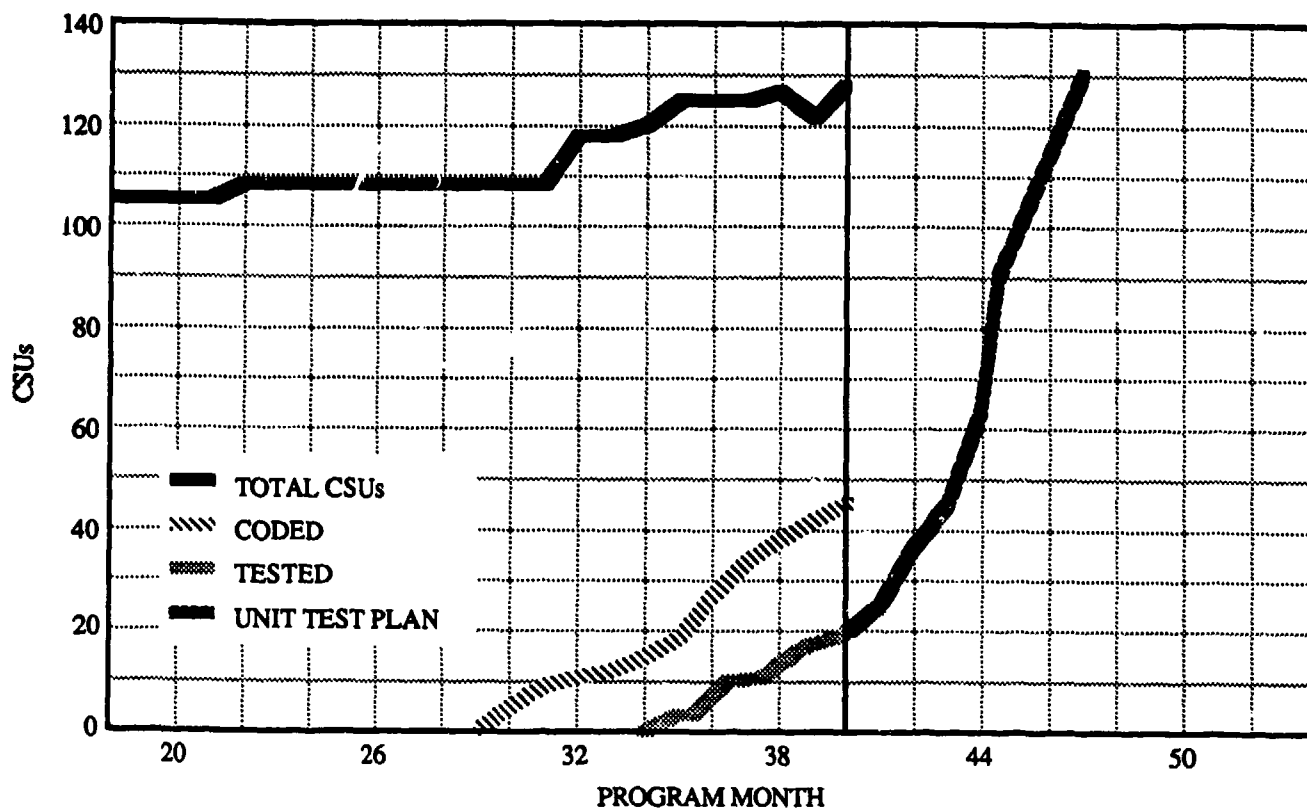


Figure 5a. Code and Test Progress

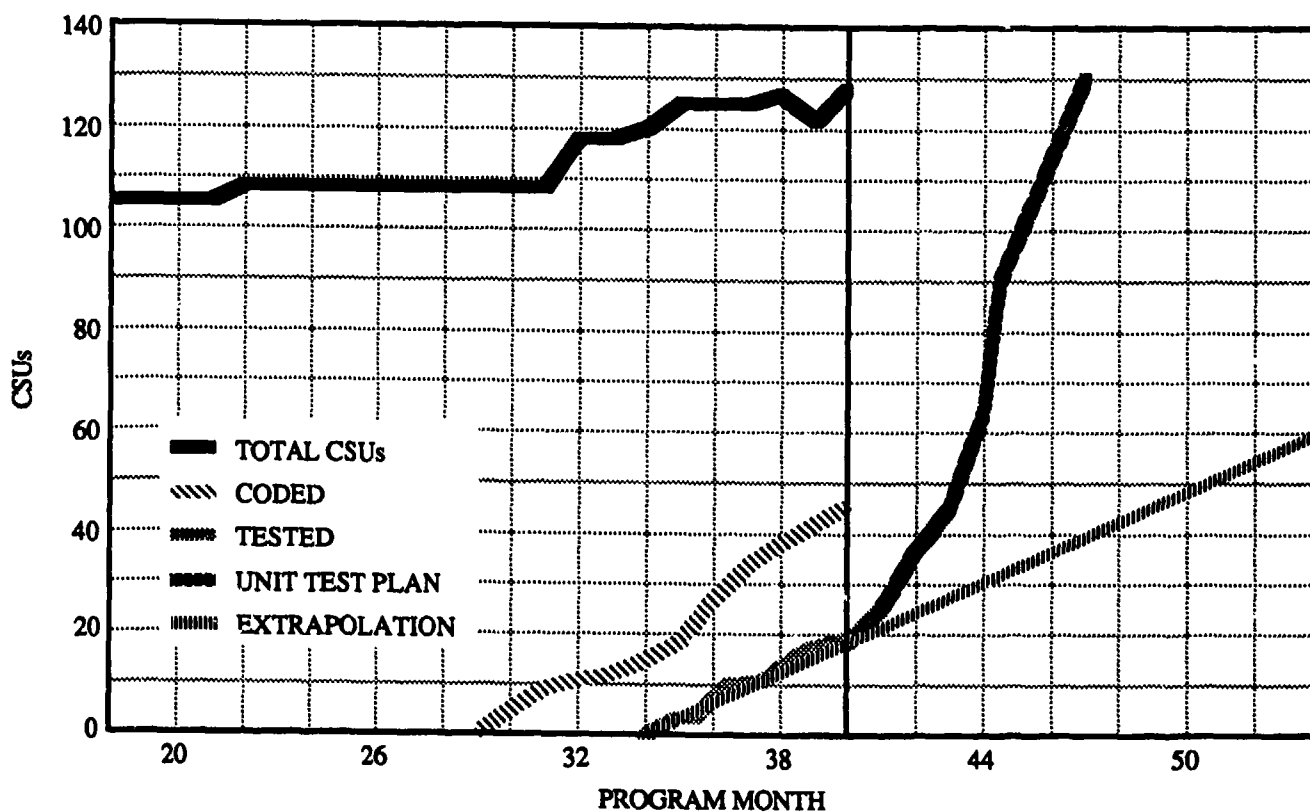


Figure 5b. Test Extrapolation

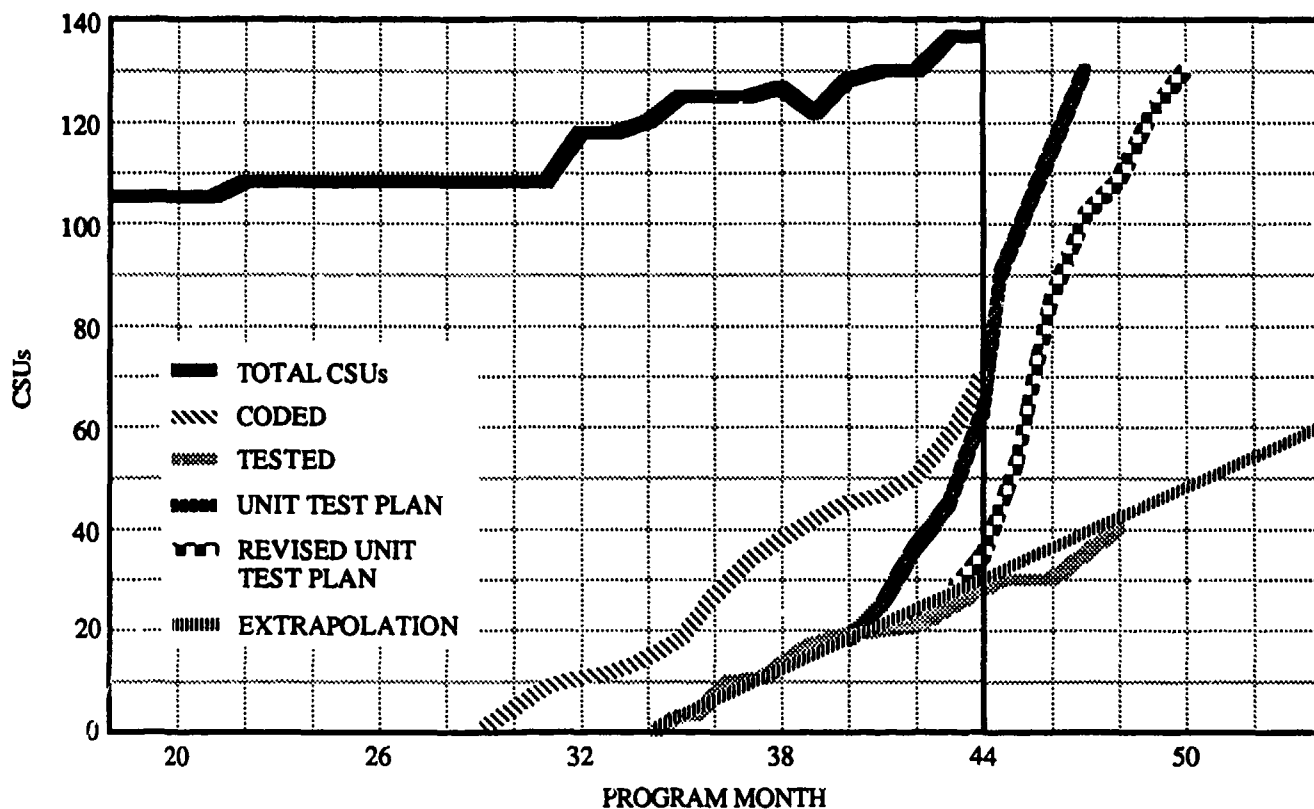


Figure 5c. Test Progress with Replan and Actuals

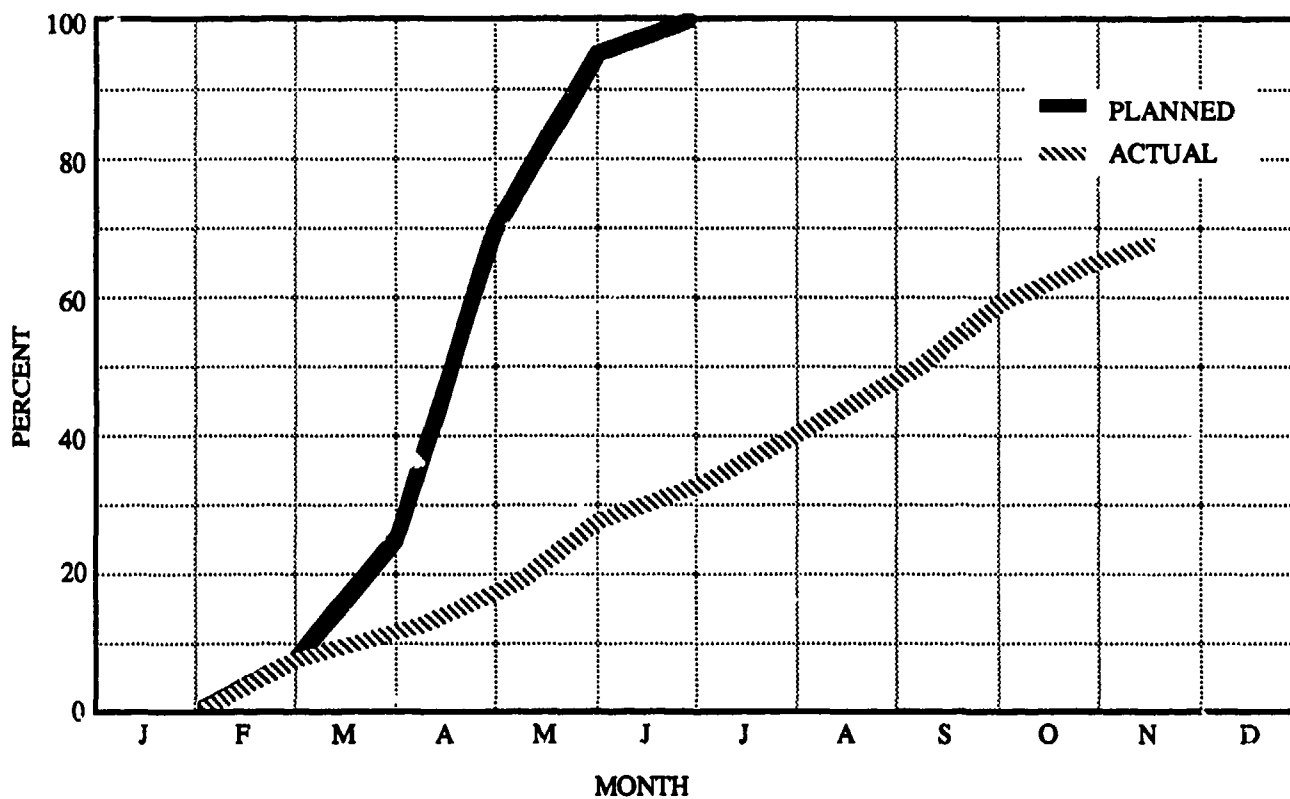


Figure 6. Number of Integration Tests

SECTION 3

The Metrics

The Software Management Metrics are described in this section. Each metric description includes a statement of purpose, typical behavior patterns, data inputs, tailoring ideas, plotting examples, and interpretation notes. The latter three require some explanation, which is provided in the following paragraphs.

Tailoring

Most of the metrics will apply to all programs. However, there are cases when an individual metric could be deleted or replaced to meet specific needs. Such tailoring also applies to the individual data items collected for each metric. Factors to consider when tailoring include the nature of the software acquisition, high-risk areas, the software metrics currently used by the contractor, the use of more than one programming language, the type of testing, and the number of Computer Software Configuration Items (CSCIs). Tailoring suggestions are included with each metric. The resulting set of data input requirements for each metric is then contractually specified in a DID referenced by the Contract Data Requirements List (CDRL). Examples of DID backup sheets for a basic set and for a tailored set of metrics are given in appendices B and C, respectively. The reader is cautioned that the tailored set is merely an example, and that each acquisition must be analyzed to determine which, if any, changes are needed and appropriate. Definitions of certain data items are provided in appendix D.

Ada and Object-Oriented Design

The use of Ada and the use of an object-oriented design methodology affect the collection of certain metrics data. Ada and object-oriented design may be used separately or in conjunction; therefore, the impli-

cations of each are noted with those metrics that would require modification.

Plotting Examples

An example plot is included for each metric. The plot format is important and should conform to certain guidelines. Each plot should present at least the past 12 months of planned and actual data and the next 5 months of plan data. Review milestones are indicated on the abscissa. Plan data is the original plan submitted to the Government. Revised plans may be added and so labeled, but previous plans may not be removed because important trend data would be lost. The current month is identified by a bold vertical line on the chart. The metrics are normally reported at PMRs and therefore have the same reporting period. The example plots are based on a software development initially estimated to require 120,000 Source Lines of Code (SLOC) and 1200 staff months over a 36-month schedule. The project milestones are System Requirements Review (SRR) at month 2, System Design Review (SDR) at month 5, Software Specification Review (SSR) at month 8, Preliminary Design Review (PDR) at month 12, Critical Design Review (CDR) at month 17, Test Readiness Review (TRR) at month 27, and Physical Configuration Audit (PCA) at month 36.

Interpretation Notes

The interpretation notes contain specific guidelines derived from experience and from analyses of actual projects. They are intended to be viewed as helpful information, not as inflexible principles. Again, the metrics must be carefully analyzed by staff and by senior managers if they are to have an impact on management decisions and on subsequent software development costs and schedules.

Software Size Metric

Purpose

The Software Size metric tracks changes in the magnitude of the software development effort. SLOC to be developed directly relates to the software engineering effort necessary to build the system. SLOC is also the primary input parameter to almost all software cost estimation models used in Mission Critical Computer Resource (MCCR) applications. An increase in SLOC estimates can lead to schedule slips and staffing problems. An increasing trend should trigger steps to counter the trend or to plan for a larger effort. SLOC count is initially an estimate, but as the design matures and code is developed, the count becomes more and more accurate until it represents the actual code at completion.

SLOC is usually tracked for each CSCI as well as for the total system. To be complete, SLOC counts for all commercial off-the-shelf (COTS) and modified off-the-shelf (MOTS) software should also be included.

Behavior

Some programs show increases in estimates of SLOC over time while others show decreases. Increases may be due to a better understanding of the requirements, a better understanding of the design implications and complexities, or an optimistic original estimate, whereas decreases usually result from an overestimate at the beginning of the program and not from changes in requirements. Both may be due to an original lack of understanding and appreciation of the requirements.

Data Inputs

Each reporting period:

- a. Estimated new SLOC -- newly developed code.

- b. Estimated reused SLOC -- existing code used as is.
- c. Estimated modified SLOC -- existing code requiring change.
- d. Estimated total SLOC -- all code (sum of above).

A definition of SLOC that both the contractor and the SPO understand and accept should be used. A recommended example taken largely from reference 1 is:

SLOC includes each source statement created by project personnel and processed into machine code. It excludes comments and unmodified utility software. It includes job control language, format statements, and data declarations. It also includes newly developed support software.

A tighter definition could be developed depending on the source code language.

(Note: An accepted measure of SLOC in Ada is to count all nonliteral semicolons (;) in each package. SLOC counts in Ada may be higher than with other languages due to the specificity and completeness of the language.)

Tailoring Ideas

- a. Delete SLOC types not applicable, i.e., new, reused, or modified.
- b. Require separate data reporting for each coding language used.
- c. Require separate reporting for each processor and/or CSCI.
- d. Report object code size.

Example Plot

Figure 7 illustrates several changes in coding effort that would not be shown if only total

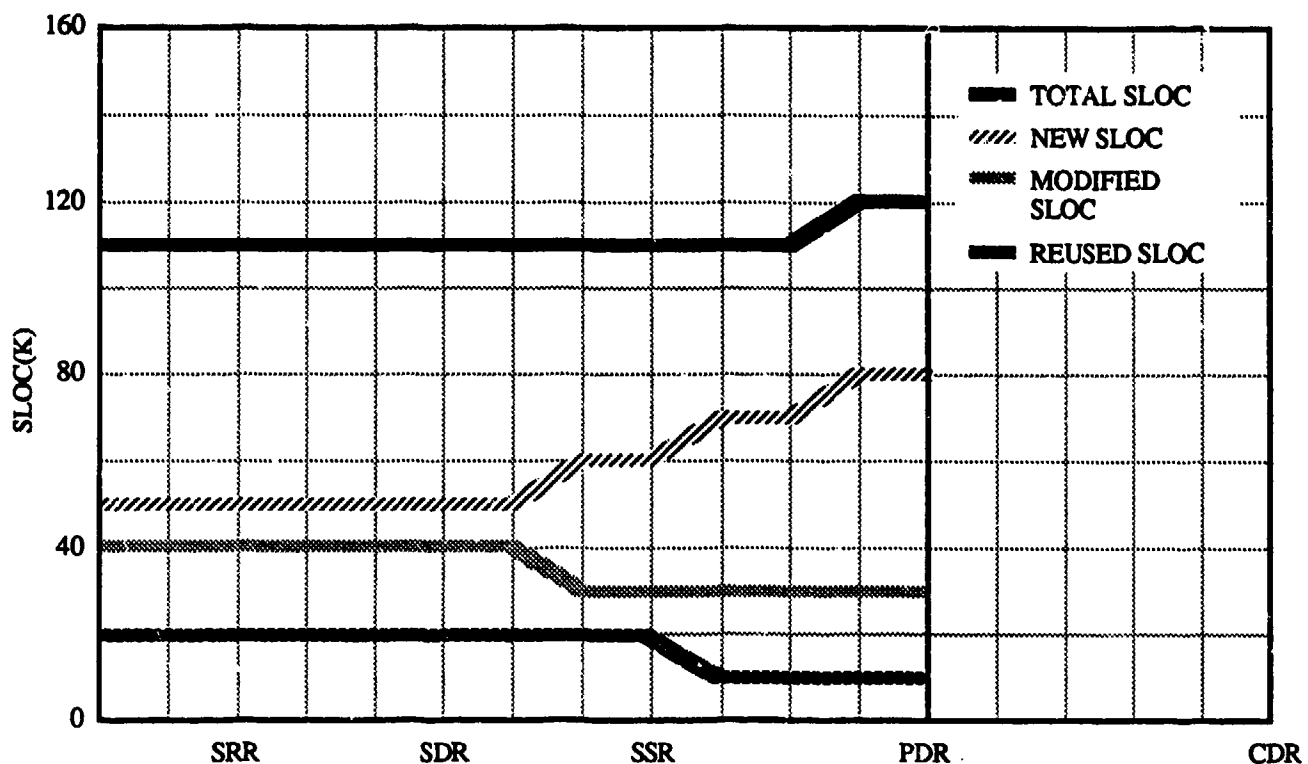


Figure 7. Software Size

SLOC were reported. A month before SSR it is determined that some modified SLOC cannot be used and that new SLOC will have to be developed. This results in an increased effort, but if only total SLOC were reported, the increased effort would not show on the graph. It is next determined that some reused SLOC cannot be used and that new SLOC will have to be developed. Again, this change requires additional effort, but if only total SLOC is reported, it again would go unnoticed. A month prior to PDR, the example shows an increase in new SLOC resulting from a better understanding of requirements. This is the only change reflected in the total SLOC count.

Interpretation Notes

- a. Software size should not vary from the previous reporting period by more than 5 percent without a detailed explanation from the contractor and related discussions regarding cost and schedule improvements.
- b. Changes in SLOC estimates often result from a better understanding of requirements, which is desirable. However, increases in size must be accounted for in the contractor's schedule and staffing plans.
- c. Total SLOC does not linearly relate to effort because modified and reused code require increasingly less effort to develop than new code. If SLOC to be modified are identified and counted at the CSU level (including SLOC that must be understood in order to modify other lines), then the modified code development effort will closely equal that for newly developed code. Similarly, reused code may also require coding effort to be integrated into a new system. Therefore, if the lines of reused code requiring modification are included in the counts of modified code, then the sum of modified and new code will approximate the total software development effort.

Software Personnel Metric

Purpose

The Software Personnel metric tracks the ability of the contractor to maintain planned staffing levels and to maintain sufficient staffing for timely completion of the program. The software staff includes the engineering and management personnel directly involved with the software system planning, requirements definition, design, coding, test, documentation, configuration management, and quality assurance. Counts of unplanned personnel losses are maintained so that work force stability can be tracked. Experienced staff are crucial to timely software development. Experienced personnel are nominally defined as those individuals with a minimum of 5 years' experience in MCCR software development and a minimum of 3 years' experience in software development for applications similar to the system under development.

Behavior

The planned 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. However, a program with too few experienced software personnel, or one that attempts to bring many personnel onboard during the latter stages of the project's schedule, will most likely experience difficulty. The normal shape of the total software staff profile is to grow through the design phases, peak through the coding and testing phases, and then gradually taper off as integration tests are successfully completed. The shape of the experienced staff profile should be high during the initial stage of the project, dip slightly during CSU development, and then grow somewhat during testing. The ratio of total to experienced personnel should typically be near 3:1 and should never exceed 6:1.

Data Inputs

Initial:

- a. Planned total personnel level for each month of the contract.
- b. Planned experienced personnel level for each month of the contract.
- c. Expected attrition rate.

Each reporting period:

- a. Total personnel.
- b. Experienced personnel.
- c. Unplanned personnel losses.

Tailoring Ideas

- a. Report staffing separately for each development task, e.g., validation and verification (V&V), support software, applications software, testing, and software quality assurance (SQA).
- b. Report staffing separately for special development skills needed, e.g., Ada, database management system (DBMS), operating system, and artificial intelligence (AI).
- c. Report staffing separately for each development organization.

Example Plot

Figure 8 shows that prior to CDR the actual number of personnel was lagging behind the plan, but that the number of experienced personnel was higher than planned. This may indicate that the contractor was initially having staffing problems and was trying to compensate by using additional experienced staff. Current levels may be appropriate if development schedules are maintained, but the SPO should monitor this closely. Unplanned losses show a nominal trend and do not indicate any internal problems. A

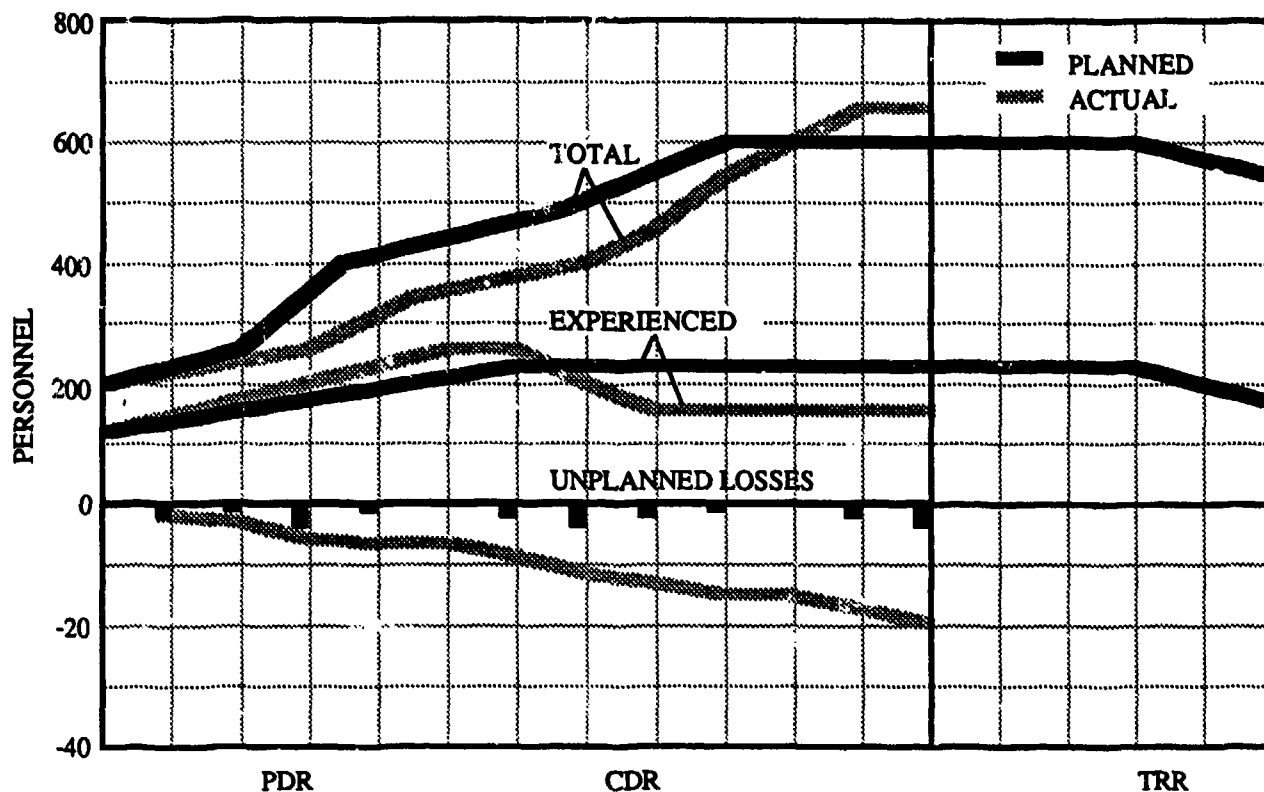


Figure 8. Software Personnel

total of 20 staff, out of an average staffing level of 400, left the project during the past 12 months.

Interpretation Notes

- Understaffing results in schedule slippage and, if not corrected, in a continuing rate of slippage. Causal relationships to various progress metrics should be examined.
- Adding staff to a late project will seldom

improve the schedule and often causes further delays.

- A program that is experiencing a high personnel turnover rate cannot maintain needed continuity. Losses that would impair the project knowledge and experience base should be discussed with the contractor.
- Initial staffing levels should be at least 25 percent of the average staffing level.

Software Volatility Metric

Purpose

The Software Volatility metric tracks changes in the number of software requirements and in the contractor's understanding of these requirements. The two graphs used for this metric track requirements changes and Software Action Items (SAIs). The graph of requirements changes tracks the total number of software requirements (typically the number of "shalls" in the Software Requirements Specifications (SRSs)) as well as the cumulative number of changes to those requirements. The graph of SAIs tracks the number of unresolved requirement/design issues. Both graphs are good indicators of requirement and design stability. Changes in the number of requirements (both additions and deletions) directly impact the software development effort. Changes are expected in the early stages as details of the system's operations are being defined and understood. At some point, however, software requirements must be frozen. The longer this takes, the greater the impact on cost and schedule.

Design reviews may have several inconsistencies between the requirements and the design or within the design itself. When this occurs, an SAI is opened. It may be closed by modifying or clarifying the design or by modifying the requirements. An SAI is defined as any discrepancy, clarification, or requirements issue that must be resolved by either the contractor or the Government.

Behavior

Changes in software requirements can be expected to be more numerous during requirements analysis and preliminary design phases. *Changes occurring after CDR may be expected to have a significant schedule impact, even if the change is the deletion of a requirement.* Therefore, the plot of cumulative changes is expected to rise more steeply prior to PDR and show a leveling off after CDR.

The plot of SAIs is expected to rise at each review and then taper off exponentially. Programs that produce clear and complete specifications will experience less of a rise at each review; and programs that have good communications among the SPO, the system engineer, and the contractor will experience a high rate of decay to the curve.

Data Inputs

Each reporting period:

- a. The current total number of requirements.
- b. The cumulative number of requirement changes to include additions, deletions, and modifications.
- c. The number of new SAIs.
- d. The cumulative number of open SAIs.

Tailoring Ideas

- a. Track the longevity of open SAIs, e.g., 0-30 days, 30-60 days, 60-90 days, and over 90 days.
- b. Track open SAIs by priority.

Example Plots

The graph of requirements changes in figure 9a shows an upward trend in the number of changes prior to PDR and a leveling off approaching CDR. The changes after PDR may result in a schedule slip for CDR because some Computer Software Component (CSC) and CSU designs in the Software Design Documents (SDDs) may have to be redone.

Figure 9b shows the number of open SAIs peaking at PDR and CDR. The steady decrease after each review indicates the contractor's ability to resolve the issues.

Interpretation Notes

a. Requirements volatility between CDR and TRR will result in schedule impacts whose extent must be determined through discussions with the contractor.

b. SAIs open more than 60 days should be closely examined. They could have significant schedule impacts, especially if they have been termed "unimportant."

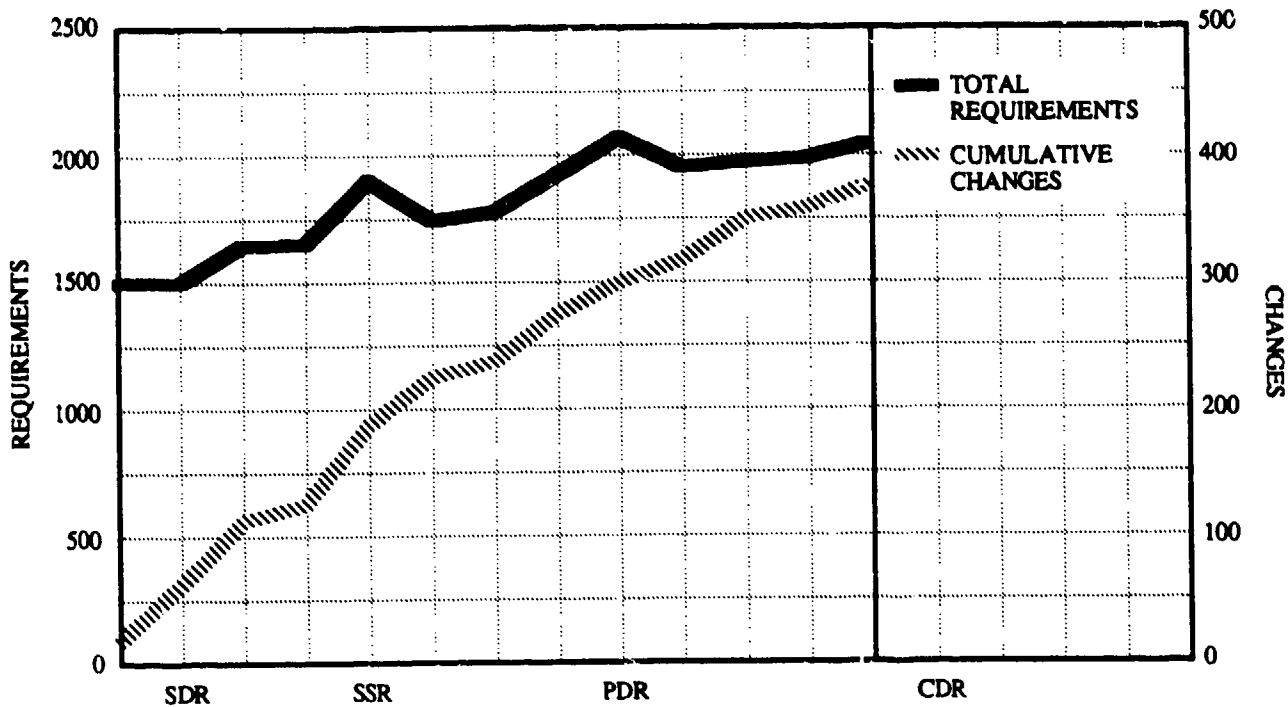


Figure 9a. Software Volatility/Software Requirements Changes

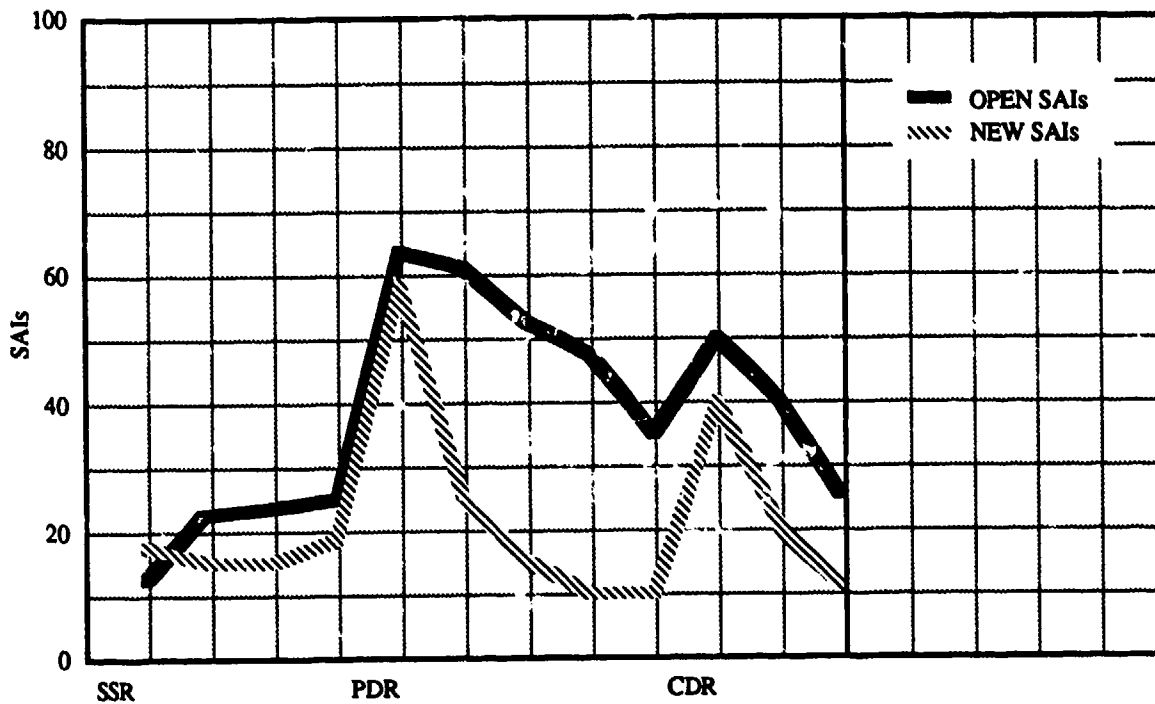


Figure 9b. Software Volatility/SAIs

Computer Resource Utilization Metric

Purpose

The Computer Resource Utilization metric tracks changes in the estimated/actual utilization of target computer resources and provides warnings if the limits of these resources are approached. Three resources typically monitored are CPU timing, memory (e.g., CPU, mass storage), and I/O channels (e.g., communications, bus). The system architecture (e.g., parallel, serial, distributed) will determine how the resources are monitored, but they must be monitored to assure that the system will fit the planned resources.

Behavior

Most projects experience an upward creep in resource utilization. Large system acquisitions typically specify a 50 percent spare capacity. This means that only one-half of the capacities may be used, leaving 50 percent for growth. If the utilization exceeds 50 percent, the project either has to expand the resource capabilities or change the system requirements. Whenever resource capabilities are expanded, the utilization curves affected will drop to new values.

Dependencies among resources result in parallel movements. For example, an expansion of memory not only decreases its utilization, but also may allow the CPU to operate more efficiently, thus decreasing CPU utilization. The same memory expansion may also allow larger blocks of data to be transported, thus reducing utilization of the I/O channels.

Data Inputs

Initial:

- a. Planned spare for each resource.

Each reporting period:

- a. Estimated/actual percentage of CPU utilization.

- b. Estimated/actual percentage of memory utilization.
- c. Estimated/actual percentage of I/O channel utilization.

Tailoring Ideas

- a. Report combined utilizations in a multi-resource architecture that uses a load-leveling operating system.
- b. Report utilizations separately in a multi-resource architecture that has dedicated functions.
- c. Report average and worst case utilizations.
- d. Report separately for development and target processors.
- e. Consider memory addressing limits of the architecture when establishing utilization limits.

Example Plot

A 50 percent spare requirement was planned for all three resources. Notice that each of the utilizations plotted in figure 10 shows a tendency to increase over time. In this example, the CPU utilization was the first to exceed the spare limit and was corrected by upgrading to a faster CPU. If growth in the same computer series is not possible, then impacts may be felt on all computer resources as well as on system and applications software. It is necessary to anticipate growth as early as possible to minimize such changes.

Interpretation Notes

- a. Performance deteriorates quickly when utilization exceeds 70 percent for real-time applications.

b. Resource expansions should be planned early in the development cycle to take into account the tendency of resource utilizations to increase over time.

c. Software development costs and schedules increase dramatically as computer resource utilization limits are approached and optimization forces design and coding changes.

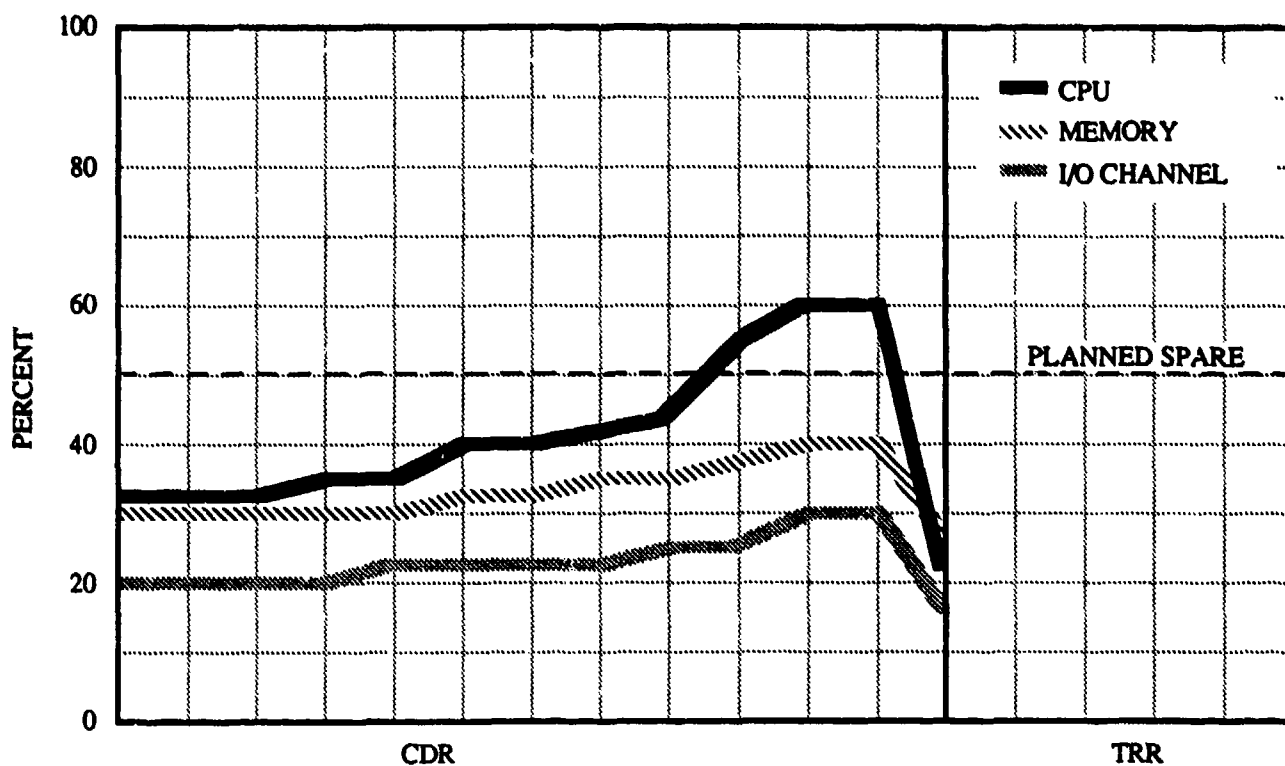


Figure 10. Computer Resource Utilization

Design Complexity Metric

Purpose

The Design Complexity metric tracks the contractor's ability to maintain an acceptable level of complexity at the CSU, CSC, and CSCI levels. A major problem in software development is the creation of highly complex designs that cannot be readily understood, and therefore are not rigorously verified through a testing process. Complex designs are also costly to maintain. The Design Complexity metric is a measure of decision structure and is calculated using a method developed by T. J. McCabe (see reference 2). If design complexity is controlled so that each CSU, CSC, and CSCI can be rigorously tested, then system testing becomes a low-risk area. It has been commonly observed that 90 percent of software problems occur in 10 percent of the code; therefore, this metric tracks those CSUs with the highest complexity, namely, the top 10 percent. If a program design language (PDL) is used, design complexity is calculated directly from the PDL logic. Appendix E defines the calculations for each complexity measure.

Behavior

A design complexity of 7 is considered the highest a CSU should have without incurring undue risk. (The exception to this is complexity resulting from the use of a Case statement.) An average complexity of 7 would not be excessive because only the top 10 percent are being reported. Because CSUs with the highest complexity may be designed later in the schedule, this metric may show an increasing trend as CSU design progresses. This trend can be correlated with the CSU Development Progress metric to determine whether the projected trend of complexity will exceed 7 before all CSU designs are completed. If so, steps can be taken to prevent this. A CSC and a CSCI should have maximum design complexities

of 40 and 400, respectively. The behavior described regarding CSU complexity applies equally to them.

Data Inputs

Each reporting period:

- a. Average design complexity of the 10 percent most complex CSUs.
- b. Average design complexity of the 10 percent most complex CSCs.
- c. Design complexity of the most complex CSCI.

(Note: For object-oriented designs, this metric will normally apply only at the CSU level. However, there are cases where it may apply to CSCs and CSCIs if the functions are so organized.)

Tailoring Ideas

- a. Track reported top 10 percent by category; e.g., the number of CSUs whose design complexity is less than 7, 8 to 20, and over 20.
- b. Track design complexity of all CSCs by category; e.g., number of CSCs whose design complexity is less than 40, 40-60, and over 60.
- c. Track design complexity of all CSCIs.
- d. Track code complexity after CDR using Halstead's or McCabe's measures [3, 4].

Example Plot

In figure 11, the plot of CSU design complexity shows an initial surge to a complexity level of 9. The contractor then took steps to reduce this by dividing overly complex CSUs into two or more CSUs.

The plot of CSC design complexity reflects the incorporation of too many complex CSUs into a CSC. This was corrected by again reducing the complexity of individual CSUs by separating them into two or more CSUs.

CSCI design complexity shows an adverse trend. A CSCI's complexity may be reduced by dividing complex CSCs within it into two or more CSCs.

Interpretation Notes

a. An error found during CSC and CSCI integration can cost 30 times more to

correct than if found at the CSU level.

b. An error found during system testing can cost 90 times more to correct than if found at the CSU level.

c. Although it has been observed that 90 percent of the errors are associated with 10 percent of the code, it has also been observed that 50 percent of the errors are in only 4 percent of the code. Therefore, if only the top 4 percent of complex CSUs could be reduced to an acceptable level, 50 percent of the potential errors might be eliminated.

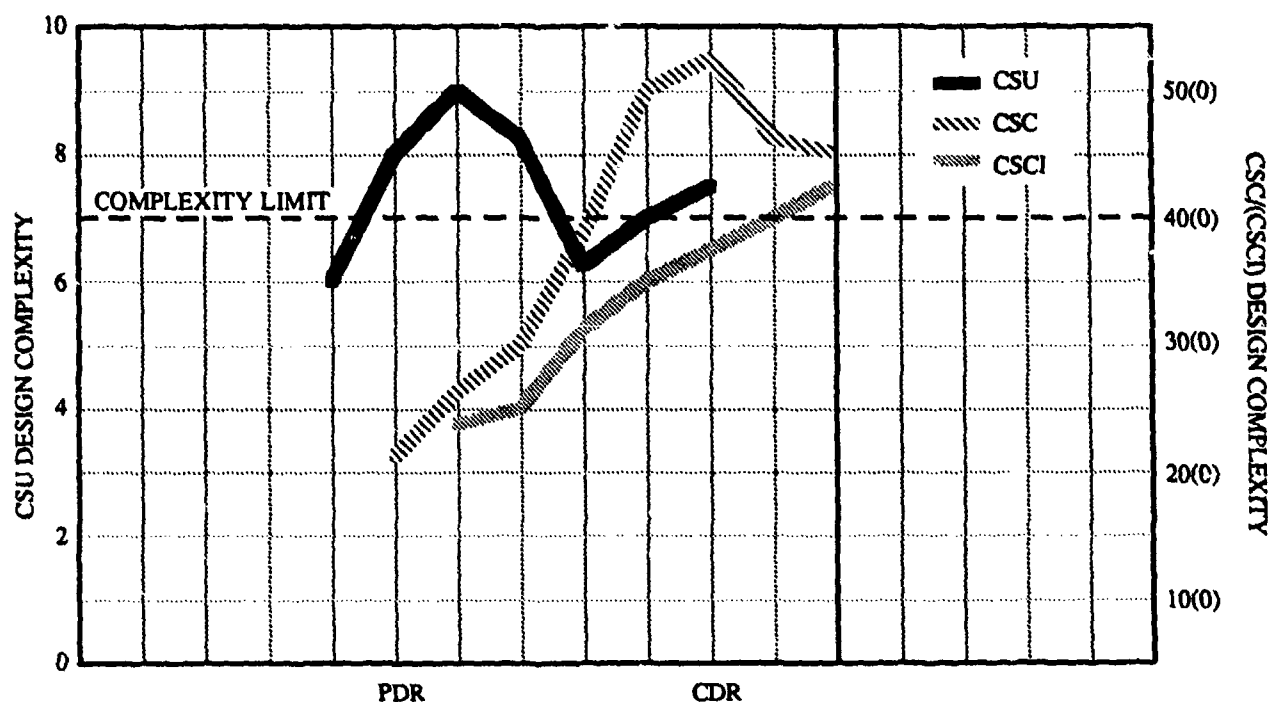


Figure 11. Design Complexity

Schedule Progress Metric

Purpose

The Schedule Progress metric tracks the contractor's ability to maintain the software development schedule by tracking the delivery of software work packages defined in the Work Breakdown Structure (WBS). The program's estimated schedule can be calculated each month by applying the relative progress to date to the program schedule. The calculation is as follows:

$$\text{Estimated Schedule (months)} = \frac{\text{Program Schedule (months)}}{\text{BCWP/BCWS}}$$

where BCWP is the budgeted cost of work performed and BCWS is the budgeted cost of work scheduled. The use of cost data to estimate progress requires close coordination with the costing staff to assure that only software costs are used in this calculation and that the contractor is credited only for work performed and confirmed by other progress metrics such as Design, CSU Development, Testing, and Incremental Release Content.

Behavior

It is not unusual for a program to initially fall behind, because insufficient time is usually allocated to the design process. This trend should level off as the design is implemented, but may again occur during testing due to inadequate test planning and inadequate testing at the CSU and CSC levels. It is important that testing at these levels is tied to WBS elements to assure visibility into its adequacy either directly or through a V&V function. Applying this metric retroactively to certain procurements running 50 to 100 percent over schedule shows the progress to be almost on schedule right up to system testing. This means that credit was given for software development progress

that had not occurred. The WBS for software must be tied to each stage of testing so that credit is not given until the adequacy and success of tests at the CSU, CSC, and CSCI levels have been verified.

Data Inputs

Initial:

- a. Number of months in program schedule.

Each reporting period:

- a. BCWP for software.
- b. BCWS for software.
- c. Number of months in program schedule (if revised).

Tailoring Ideas

- a. Track progress separately for each CSCI.

Example Plot

At month 2, the plot in figure 12 indicates that given the current rate of productivity, it will require 45 instead of 36 months to complete the project. The ratio of work performed to work scheduled was 80 percent, resulting in an estimated schedule of 45 months ($36/.8 = 45$). During succeeding months productivity dropped to around 70 percent, resulting in an estimated schedule duration of about 51 months. At this point, almost halfway through the original 36-month schedule, the schedule was revised and extended to 45 months. This change does not affect computation results because increasing the schedule by 25 percent also reduces the work by 20 percent, so the estimated schedule months remain the same. However, the plot now indicates a slip of only 6 months versus 15 months from the original schedule.

Interpretation Notes

- The plot of this metric can be used to identify and extrapolate trends. If the trend is up, it implies a worsening condition. An extrapolation can be made to predict the estimated schedule by extending the extrapolation until its value intersects with the same value on the abscissa. This is the estimated schedule based on the trend.
- If the trend is down, it indicates that productivity is under control and improving, and that the overall schedule can be predicted by extrapolating this plot and again seeking the intersection of the plot value with the abscissa.

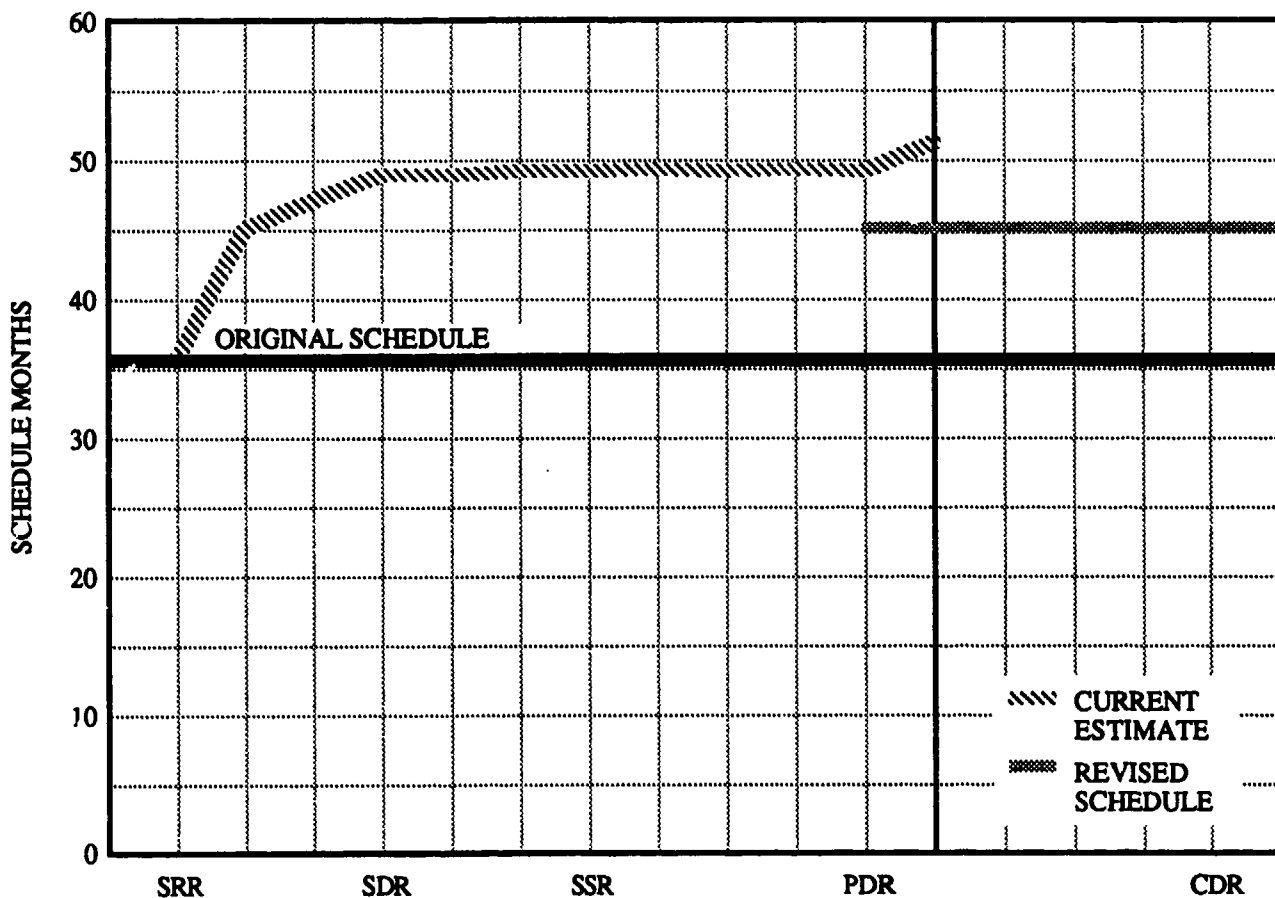


Figure 12. Schedule Progress

Design Progress Metric

Purpose

The Design Progress metric measures the contractor's ability to maintain progress during the initial software development phases. It tracks the development of the Software Requirements Specifications (SRSs) during the system design and software requirements analysis phases, and the development of the preliminary design portion of the Software Design Documents (SDDs) during the preliminary design phase. Each of these documents culminates in a review — Software Specification Review (SSR) for the SRSs, and Preliminary Design Review (PDR) for the preliminary design portion of the SDDs. Tracking the progress of the SRSs and SDDs also provides visibility into the contractor's ability to hold SSR and PDR on schedule.

Behavior

System/Segment Specification (SSS) requirements are initially allocated between hardware, software, and personnel in the System/Segment Design Document (SSDD) at System Design Review (SDR). The software requirements in the SSDD are then allocated to CSCs that are each documented in an SRS. Shortly after System Requirements Review (SRR) the contractor should establish a schedule for developing each SRS. This schedule should indicate completion 1 to 2 months prior to SSR. Similarly, the SRS requirements are allocated to CSCs and documented in the preliminary design version of the SDDs. This activity should be scheduled for completion 1 to 2 months prior to PDR. Development of the SRSs and the SDDs may fall behind the plan if system requirements are not clearly defined. Clarifications may generate SAIs (another metric) whose open status may also lead to delays in scheduled reviews.

Delays during these phases can have both positive and negative effects on the balance

of the project schedule. If delays in SSR and PDR result in clear and precisely defined requirements, then improvements in the balance of the schedule may more than offset those delays. However, if delays in SSR and PDR still do not result in well-defined requirements, then more and longer delays can be expected throughout the project, with a corresponding high incidence of SAIs.

Data Inputs

Initial:

- a. Number of SSDD software requirements to be documented in each SRS each month.
- b. Number of SRS requirements to be documented as CSCs in the SDDs each month.

Each reporting period:

- a. Actual number of SSDD software requirements completely documented in SRSs.
- b. Actual number of SRS requirements completely documented as CSCs in the SDDs.

Tailoring Ideas

- a. Track the development of the SDDs during the detailed design phase; i.e., the allocation of CSC requirements to CSUs. Completion should be scheduled 1 to 2 months prior to CDR.

Example Plot

In figure 13 the actual number of requirements documented in the SRSs is shown to fall behind the plan, but a trend developed early and provided a good basis for planning. Based on this slip, the SSR was delayed 2 months. Due to the quality of the SRSs produced, however, it was possible to begin

development of the SDDs a month ahead of the rescheduled SSR date, resulting in only a 1-month overall schedule slip. It also appears that due to the SRSs' quality, the SDDs are being developed at the scheduled rate, although a month behind schedule.

Interpretation Notes

a. If the deviation between planned and actual data is increasing, it implies that the further into the requirement speci-

fication the contractor goes, the more confusing, inconsistent, unclear, and/or untestable the requirements become. This implies major problems with the SSDD. No approval to proceed beyond SSR should be given until a good SRS is produced and clearly understood.

b. If the deviation between planned and actual data is decreasing, it implies that requirement problems may already have been addressed and that the completion of this task can be projected.

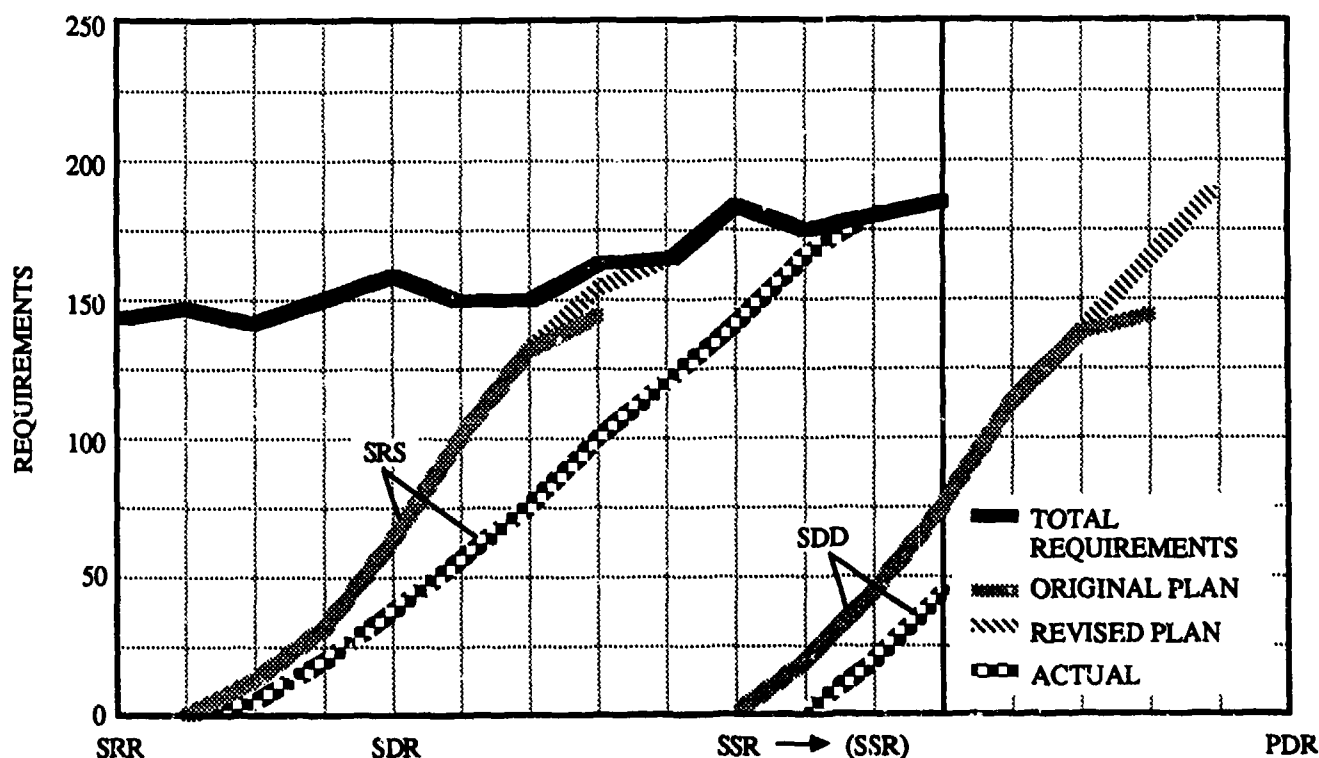


Figure 13. Design Progress

CSU Development Progress Metric

Purpose

The CSU Development Progress metric tracks the contractor's ability to keep CSU design, coding, test, and integration activities on schedule. After CDR, the next formal milestone that indicates the status of software development is Test Readiness Review (TRR). This is too late in a program to recover if significant problems are not discovered until then. This metric provides visibility into development progress throughout the development phase.

Behavior

The design of CSUs usually begins around PDR. A count is plotted of the number of CSU designs that have passed internal review. After CDR, when the contractor begins coding, a healthy program will experience a steady progression of CSU test completions. Sporadic CSU development can be caused by factors such as overutilized development computers or under-experienced staff. This results in a high-pressure environment in which all software becomes due at once. For a well-run program, the plots of CSUs designed, tested, and integrated should each rise with a fairly constant slope.

The CSU design, test, and integration schedule should be reported by the contractor at CDR. The number of CSUs whose design packages have been closed, the number of CSUs passing CSU test and placed under configuration control, and the number of CSUs integrated will be updated at the end of each calendar month of the contract.

Data Inputs

Initial:

- a. Number of CSUs to be designed each month.
- b. Number of CSUs to be tested each month.

- c. Number of CSUs to be integrated each month.

Each reporting period:

- a. Actual number of CSUs designed.
- b. Actual number of CSUs tested.
- c. Actual number of CSUs integrated.

(Note: Ada CSU design progress may be measured by counting the number of packages whose specifications are complete. This is using Ada as a PDL and measuring CSU (package) design progress to completion. For object-oriented designs, CSU integration progress may be tied to an integration plan rather than to CSCs.)

Tailoring Ideas

- a. Plot very large CSCIs separately or plot each CSCI when there are only two or three in the system.
- b. Track CSU development progress by CSCs.
- c. Track number of CSUs for which PDL code has been developed.
- d. Report defect density from errors found at CSU design inspections, i.e., average number of defects per CSU.
- e. Report defect density from errors found at CSU test inspections, i.e., average number of defects per CSU.

Example Plot

In figure 14, the plot of CSUs actually designed follows closely the plot of CSUs planned to be designed until 3 months after PDR. At this point, the actual number begins to deviate significantly from the planned number. The contractor's explanation was that changing requirements prolonged the design of the system.

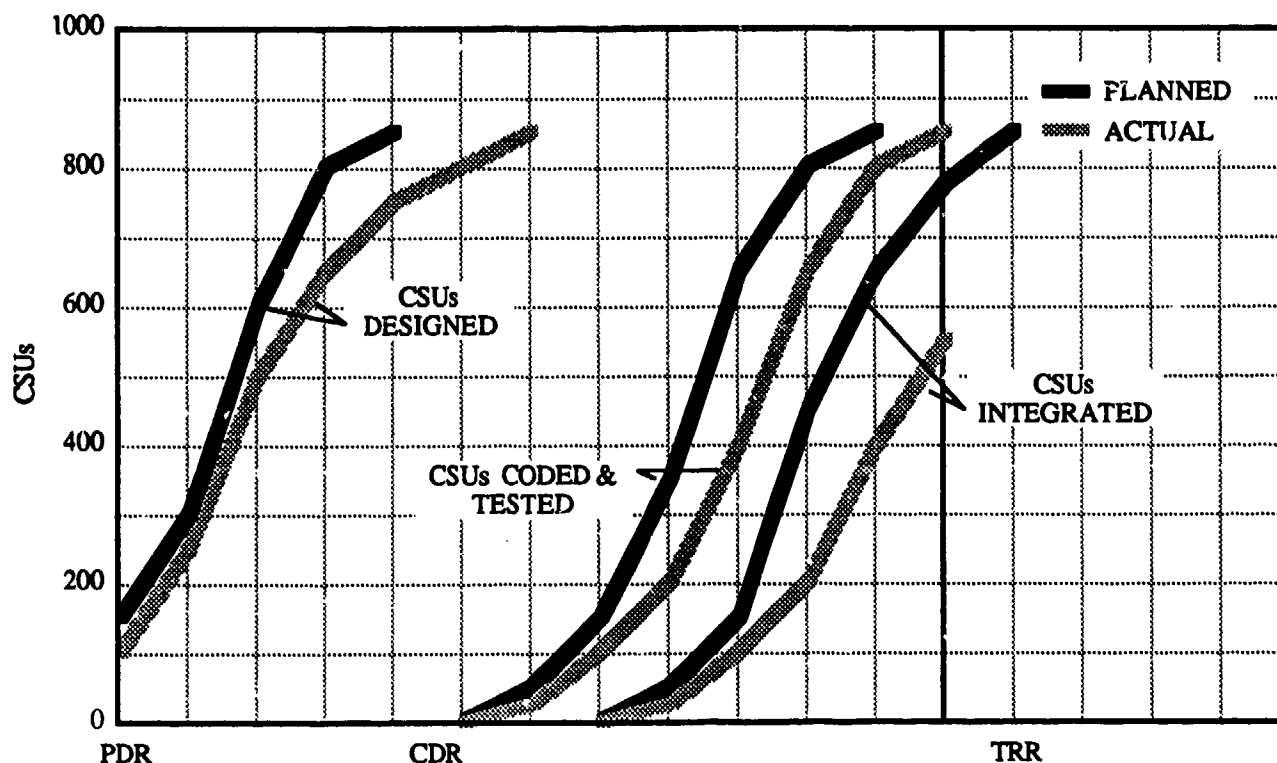


Figure 14. CSU Development Progress

The number of CSUs actually tested and integrated lags behind the plan. The SPO ascertained from the contractor that the lag resulted from an increased number of 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 effect any possible schedule improvements with the contractor.

Interpretation Notes

Wide variations in schedule profiles have been followed by different projects. The following general information should not be misused through overly specific interpretation. It is offered on the premise that some guidance is better than none. For more data, see references 1 and 5. Contributors to wide

variations in schedule performance are the availability and use of modern software development tools such as PDLs, automated debugging facilities, and database tools.

a. Schedule for Software Development

The number of calendar months (M) from the beginning of software design to the end of PCA 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, system tested, and documented.) The following formulas have been empirically derived from a database of ESD/MITRE projects. The formulas describe curves for three project percentile values. For example, only 5 percent of

the projects in the Construction Cost Model (COCOMO) database were able to preserve a schedule equal to or shorter than the schedule predicted by $M = 3^{*3}\sqrt{SM}$.

	Percentile
$M = 3^{*3}\sqrt{SM}$	5%
$M = 3.8^{*3}\sqrt{SM}$	50%
$M = 4.6^{*3}\sqrt{SM}$	95%

b. Source Lines of Code per Staff Month

Typical SLOC/SM values are:

- 150 for easy code
 - 70 for moderately difficult code (C³I system)
 - 30 for difficult code (e.g., security)
- (Note: SM refers to the number of software

staff months expended from contract award through PCA.)

c. Allocation of Staff and Schedule for Software Development

These are desired values based on experience with Mission Critical Computer Resource (MCCR) software programs similar to the C³I systems developed at ESD. Wide variations from these values have been observed, but experience indicates that increases in design effort lead to reduced integration and test effort and to higher quality products.

	Staff Months	Schedule
Design	45%	50%
CSU Code and Test	20%	15%
Integration and System Test	35%	35%

Testing Progress Metric

Purpose

The Testing Progress metric measures the contractor's ability to maintain testing progress and the degree to which the software is meeting system requirements. Two separate graphs are used for this metric. The test status graph tracks the progress of CSCI and system testing against a plan. A graph of Software Problem Reports (SPRs) focuses attention on the number of open software problems and the density of discovered errors.

The test status graph shows separate plots for CSCI formal qualification tests and for system tests. CSCI testing begins after TRR, and system testing begins at the conclusion of CSCI testing and is completed prior to PCA.

The SPR graph plots the number of new SPRs reported each month and the current number of open SPRs. Also plotted is SPR density, which is the cumulative number of SPRs per 1000 SLOC. A separate set of SPR plots is shown for problems identified during CSCI and system testing.

Behavior

The test status graph shows the planned and actual counts of CSCI and system tests. The plot of tests completed should overlay the plot of tests scheduled if all tests are passed on schedule. However, test programs experience schedule slips or failed tests. Therefore, the plot of tests completed can be expected to fall below the plot of tests scheduled. The extent of this difference and the trend it proscribes are indicators of the readiness of the CSCIs and/or the system for testing and of the time it will take to complete testing.

The SPR graph provides a more detailed view of testing progress. If the number

of open SPRs from CSC testing is not approaching zero as TRR approaches (or as PCA approaches for CSCI and system testing), then these milestones will have to be delayed. Schedule slips may be estimated by observing the trend of open SPRs. An ambitious test plan may prevent the number of open problems from decreasing until all tests have been performed. This is to say that new problems may be generated faster than old ones can be resolved.

Data Inputs

Initial:

- a. Planned number of CSCI tests to be completed during each month of the contract.
- b. Planned number of system tests to be completed during each month of the contract.

Each reporting period:

- a. Number of CSCI tests passed.
- b. Number of system tests completed.
- c. Number of new SPRs.
- d. Cumulative number of open SPRs.
- e. SPR density: cumulative number of SPRs per 1000 SLOC.

(Note: If the design is object-oriented and/or is not organized by CSCs and CSCIs, then testing progress must be tracked against functionality tests described in both the integration and system test plans.)

Tailoring Ideas

- a. Track the longevity of open SPRs. For example, track the number open 0-30 days, 30-60 days, 60-90 days, and over 90 days.

- b. Track open SPRs by priority, e.g., high, medium, and low; or critical for integration, critical for initial operation, or other.
- c. Track open SPRs by type of software affected, e.g., application, operating system, and support test.
- d. Report SPRs by open, assigned, pending, resolved, rejected, and/or closed.
- e. Report SPR density by category; e.g., the number of CSUs whose SPR density is 0-10, 11-20, 21-30, and over 30.

Example Plots

In the example test status graph shown in figure 15a, the plot of tests completed falls

behind those scheduled. It can be expected that this trend will continue unless the contractor can increase the rate of testing by increasing the number of qualified personnel conducting tests or the number of hours spent each day performing tests, and neither of these is likely. This means that the start of system testing will be delayed and that its rate of progress will probably be similar to that for CSCI testing.

The SPR graph in figure 15b shows a separate plot for SPRs occurring prior to CSCI and system testing. The TRR was rescheduled because there were too many open SPRs. The SPR density ranges between 8 and 20 SPRs/1000 SLOC, which is within the normal range.

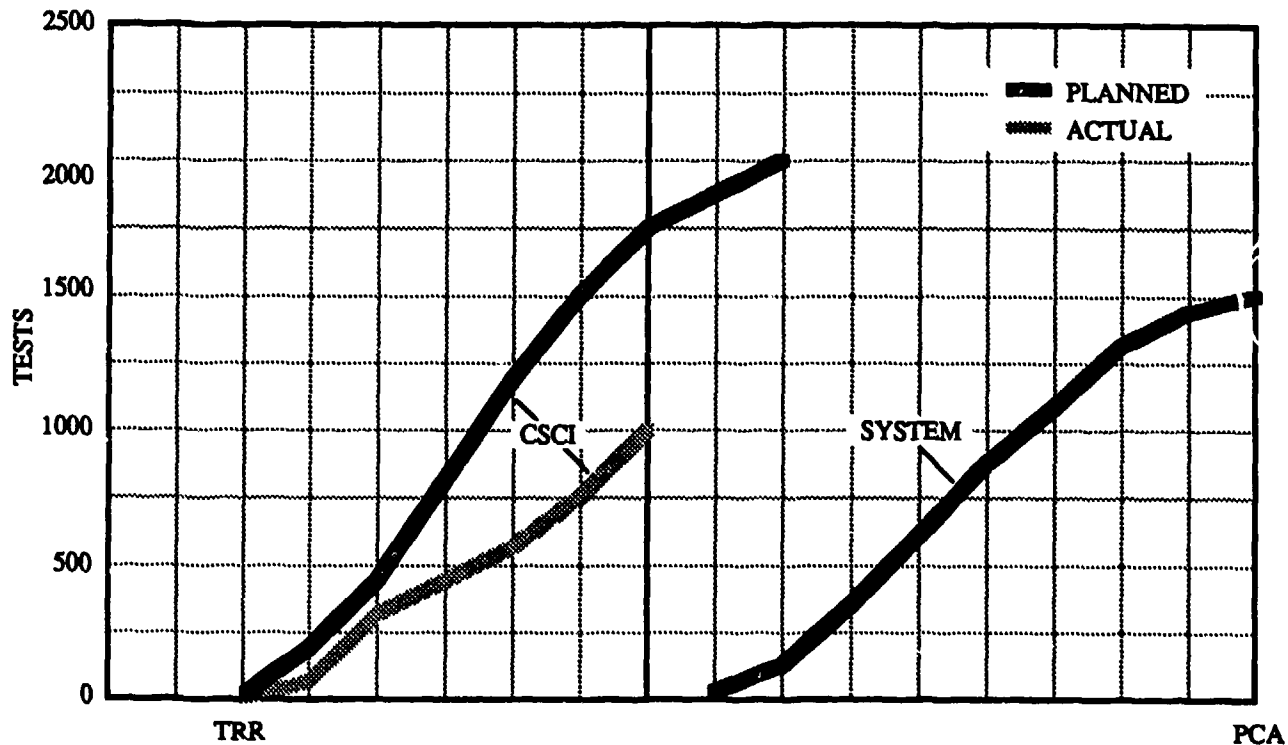


Figure 15a. Testing Progress/Test Status

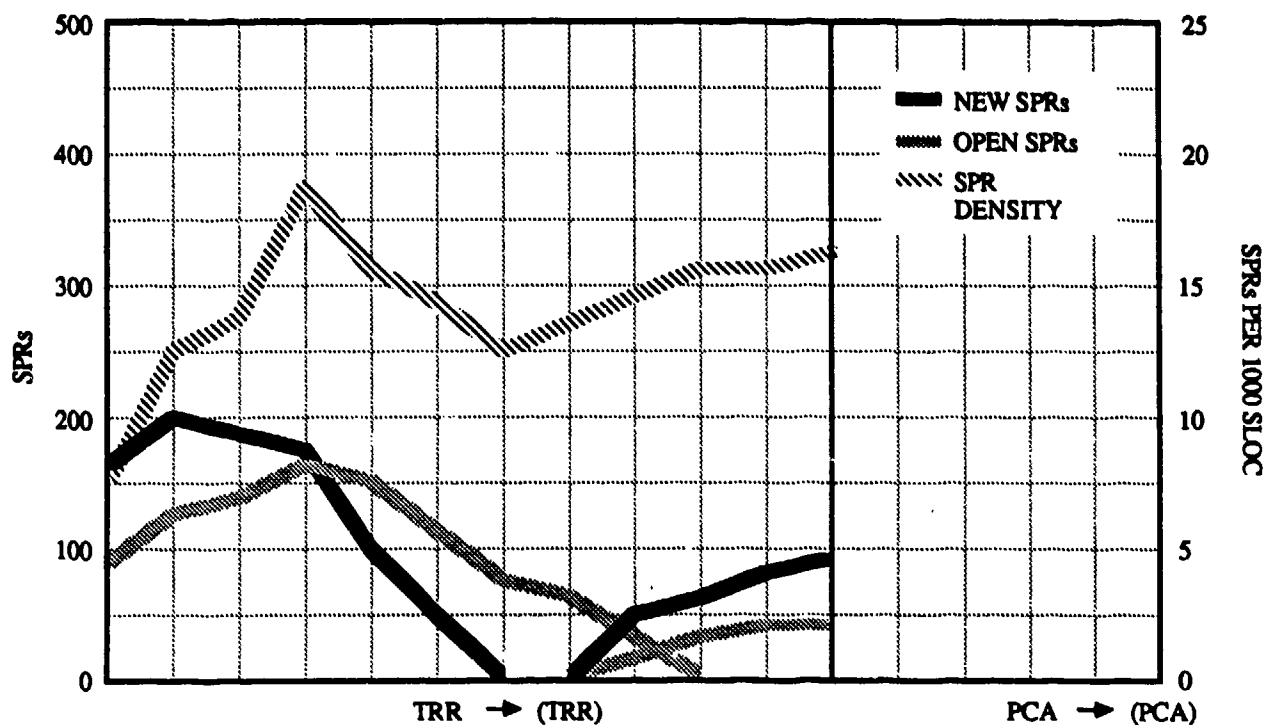


Figure 15b. Testing Progress/Software Problem Reports

Interpretation Notes

- a. The number of SPRs/1000 SLOC is an indication of testing adequacy and of code quality. A normal range may be between 5 and 30 SPRs/1000 SLOC, with 10 to 20 a safer range. Too few SPRs may indicate poor testing and too many may indicate poor code quality. In

either case, the code may still contain a large number of undetected errors.

- b. If the plot of open SPRs has a positive slope, it indicates that problems are being identified faster than they can be resolved. If the slope is negative, then the resolution of problems can be projected.

Incremental Release Content Metric

Purpose

The Incremental Release Content metric tracks the schedule and the number of CSUs per release in order to monitor the contractor's ability to preserve schedule and functionality in each planned release. *This metric also applies to software builds.* A common approach used to preserve schedule is to postpone functional capability. Decreases in the number of CSUs per release may indicate an off-loading of functions from early releases to later releases. Increases in the number of CSUs per release may indicate that a program is having unanticipated growth in the complexity of the functions to be delivered.

Behavior

Ideal behavior is for the number of CSUs in each release and the schedule dates to remain unchanged. However, typical behavior is for the number of CSUs in early releases to decrease and the number in later releases to increase as the release dates approach. This postponement is exacerbated by code growth, which also increases the number of CSUs in later releases. In a program whose designs were prepared accurately and completely for PDR and CDR, the number of CSUs should not increase significantly after CDR. A program being developed on schedule will implement the planned number of CSUs in each release.

The planned number of CSUs to be included in each anticipated release is updated at the end of each month of the contract, but the original plan always remains on the graph. The original plan for each release extends to the release date.

(Note: The term "release" does not necessarily refer to contractually obligated delivery of software. The normal software development process, in which the contractor assem-

bles 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.)

Data Inputs

Initial:

- a. Planned number of CSUs to be included in each release.
- b. Planned completion date for each release.

Each reporting period:

- a. Current number of CSUs to be included in each release.
- b. Current completion date for each release.

Tailoring Ideas

- a. Track the number of CSCs in each release as another measure of the functionality included in each release.
- b. Track the number of requirements satisfied in each release.

Example Plot

Figure 16 shows content and schedule: a two-dimensional tracking of planned releases. Shortly after CDR the number of CSUs included in release 1 is reduced from 200 to approximately 150. This means that 250 CSUs will have to be integrated for release 2; not the 200 originally planned. The number of CSUs in release 2 is reduced the following month from 400 to 350, thereby bringing the increment between the two builds back to the 200 CSUs originally planned. In the meantime, the number of CSUs in release 3 has been increasing due to an increase in total software size. At the point where release 2A begins, the number

of CSUs to be integrated for release 3 has grown to almost 500. It is obvious to the contractor that an additional release is needed; hence, release 2A is inserted, which integrates 300 of the 500 CSUs, leaving approximately 200 for release 3.

Another solution not shown in figure 16 would be to reschedule the completion dates for each release. However, this is less acceptable to both the contractor and the Government because each schedule slip attracts management's attention. In this example, however, the productivity rates implied by the progress to date indicate that

the scheduled completion dates for releases 2A and 3 will likely be slipped. This likelihood should be discussed with the contractor.

Interpretation Notes

- The number of CSUs per release should remain reasonably stable. However, a small increase in the number of CSUs can be expected as the program's design matures.
- Builds should be encouraged that correspond to operationally useful capabilities as soon as it is practicable.

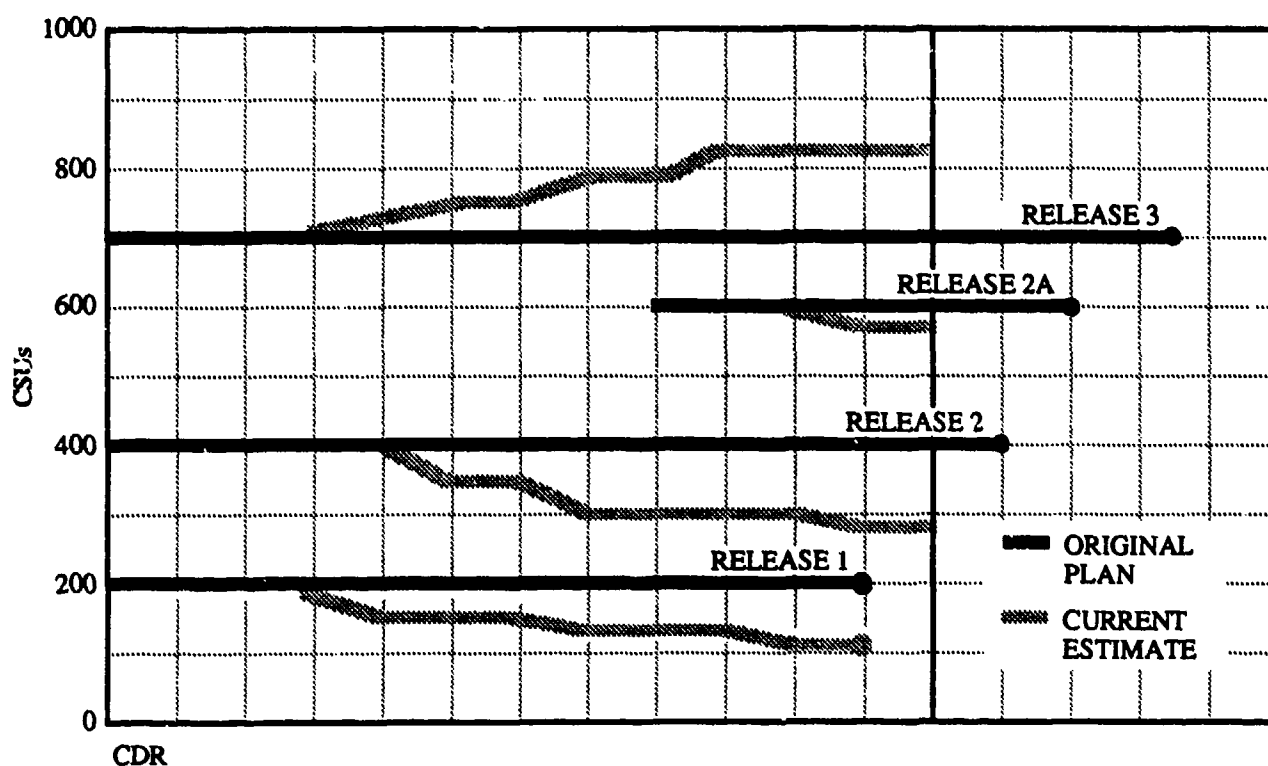


Figure 16. Incremental Release Content

List of References

1. Boehm, Barry W., *Software Engineering Economics*, Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

This book describes the operation and philosophy behind one of the important software cost estimating models in use today. It discusses factors that impact the cost and schedule of software developments.

2. McCabe, T. J., et al., "Structured Testing," 14th edition, McCabe and Associates, Columbia, Maryland, September 1987.

3. Halstead, M. H., *Elements of Software Science*, New York: Elsevier, 1977, pp. 274-279.

4. McCabe, T. J., "A Complexity Measure," *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, December 1976, pp. 308-320.

5. Albrecht, A. J., and J. E. Gaffney, Jr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering*, Vol. SE-9, No. 6, November 1983, pp. 639-648.

Bibliography

Beizer, B., *Software System Testing and Quality Assurance*, New York: Van Nostrand Reinhold, 1984.

Conte, S. D., H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models*, Menlo Park, California: Benjamin/Cummings, 1986.

DeMarco, T., *Structured Analysis and System Specification*, Englewood Cliffs, New Jersey: Prentice-Hall, 1978.

Mills, H. D., *Software Productivity*, Boston, Massachusetts: Little, Brown and Co., 1983.

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.

Turner, J., "The Structure of Modular Programs," *Communications of the ACM*, Vol. 23, No. 5, May 1980, pp. 272-277.

APPENDIX A

General Software Acquisition Information

The metrics described in Section 3 are accompanied by notes that relate to each metric. This appendix contains phase-specific information applicable to many software acquisitions. The information was obtained from software acquisition personnel and from references 1 and 5, and is intended to provide general guidance. Exceptions to these guidelines may be appropriate for specific programs; but when an exception is made, it is important to have a clear understanding of the justification.

1. Pre-Award

a. Resource Availability

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

(1) Large software development projects should use modern methodologies with automated support tools for software development. Tools should be available for such functions as static analysis, testing, design, measurements, and configuration management.

(2) Sources of experienced software engineering personnel must be identified and available during the designated software development time period.

b. Requirements Definition

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

c. Contract Preparation

The Instructions for Proposal Preparation (IFPP), Statement of Work (SOW), and CDRL should include requirements for the contractor to deliver the Software Management Metrics graphs and data.

2. Source Selection

- a. Difficult and high-risk functions should receive early and adequate attention. Workable solutions should be demonstrated before start of full-scale development.
- b. The software development schedule must 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.
- c. 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.
- d. The Government should assure that any software development by subcontractors is closely managed by the prime contractor. This is best accomplished by collocation.
- e. 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.

3. Preliminary and Critical Design Reviews

- a. Development phases should not be allowed to proceed faster than the achieved ground work can support. For example, the design should not advance beyond the status of requirements definition, coding beyond what has been designed, and testing beyond the stability of the product.
- b. Monthly or bimonthly software 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.

4. Test and Integration

- a. Software should be tested by a separate and independent organization from that which developed the software.
- b. The formal software acceptance testing schedule should allow time and access to the equipment for Government personnel to exercise the system.
- c. Integration test planning should be completed early enough to ensure that test issues can influence design, and allow long-lead-time test facilities to be acquired. For most programs, test planning should start before PDR.

APPENDIX B

Sample DID Backup Sheet

Possible uses:

DI-A-7089 Conference Minutes
DI-MGMT-80227/T Contractor Progress,
Status, and Management Report

The contractor shall substitute or add the following to Blocks 3 and 10 of the DID:

Block 3: to provide the USAF with insight into the software development progress, status, and problem areas.

Block 10: replace section 10 with the following:

The Software Management Metrics are a high-level summary of the status of the software development effort. Specific guidelines for their contents are given in the paragraphs below.

In the context of this DID, the term "current reporting period" shall refer to the interval of time between the past (i.e., most recent) and present submission dates. The frequency of submission for the report is provided in the CDRL item for this DID.

The contractor shall provide data for the metrics listed below in both graphic and tabular form. This information shall be updated as necessary for the current reporting period. The metrics data shall be provided in graphics form at the contract level for each contractor/subcontractor. Graphs shall also be prepared at the CSCI level for certain metrics as designated below. Additional graphs shall be provided as required to address high-risk or problem CSCIs. Graphs show 12 months of history and 5 months of projections where appropriate. *Revised plans may be added to the graphs, but previous plans including the original plan shall not be removed.*

Software Size

For the current reporting period, provide in

tabular and graphic form for each CSCI an estimate of:

1. Newly developed Source Lines of Code (SLOC);
2. Reused SLOC — existing code used as is;
3. Modified SLOC — existing code requiring change; and
4. Total SLOC — all code (sum of above).

SLOC includes each source statement created by project personnel and processed into machine code. It excludes comments and unmodified utility software. It includes job control language, format statements, and data declarations. It also includes newly developed support software.

Software Personnel

Provide a plan that includes, for each month of the contract:

1. The planned number of software personnel, and
2. The planned number of experienced software personnel.

For the current reporting period, provide actual counts of:

1. The total number of software personnel,
2. The number of experienced personnel, and
3. The number of unplanned personnel losses.

Software personnel include engineering and management personnel directly involved

with software system planning, requirements definition, design, coding, test, integration, documentation, configuration management, and quality assurance. Experienced personnel are defined as those with over 5 years of experience, of which 3 years are with systems similar to the one under development.

Software Volatility

For the current reporting period, provide:

1. The current total number of software requirements,
2. The cumulative number of requirements changes,
3. The number of new Software Action Items (SAIs), and
4. The cumulative number of open SAIs.

An SAI is defined as any discrepancy, clarification item, or requirement issue that must be resolved by either the contractor or the Government.

Computer Resource Utilization

For the current reporting period, provide estimated percentage utilizations of:

1. CPU computation power for each CPU in multi-CPU configurations,
2. On-line memory, and
3. I/O channel capacity.

Design Complexity

For the current reporting period, provide:

1. The average design complexity of the 10 percent most complex CSUs,
2. The average design complexity of the 10 percent most complex CSCs, and
3. The design complexity of the most complex CSCI.

(Note: See attached for definition and discussion of design complexities. (This discussion is contained in appendix E of this document.))

Schedule Progress

Initially provide the number of months in the program schedule.

For the current reporting period, provide:

1. The budgeted cost of work performed (BCWP) for software,
2. The budgeted cost of work scheduled (BCWS) for software, and
3. The number of months in the program schedule (if revised).

Design Progress

Provide a plan that includes, for each month of the contract:

1. The number of System/Segment Design Document (SSDD) software requirements to be allocated and documented in Software Requirement Specifications (SRSs) each month, and
2. The number of SRS requirements to be allocated and documented as CSCs in Software Design Documents (SDDs) each month.

For the current reporting period, provide:

1. The number of SSDD software requirements completely documented in SRSs, and
2. The number of SRS requirements completely documented as CSCs in SDDs.

CSU Development Progress

Provide a plan that includes, for each month of the contract:

1. The number of CSUs to be designed,
2. The number of CSUs to be tested, and
3. The number of CSUs to be integrated.

For the current reporting period, provide actual counts of:

1. The number of CSUs designed,
2. The number of CSUs tested, and
3. The number of CSUs integrated.

Testing Progress

Provide a test plan that includes, for each month of the contract:

1. The number of CSC tests,
2. The number of CSCI tests, and
3. The number of system tests scheduled to be performed.

For the current reporting period, provide:

1. The cumulative number of CSC, CSCI, and system tests passed;
2. The number of new software problem reports (SPRs);
3. The cumulative number of open SPRs; and

4. The cumulative number of SPRs per 1000 SLOC.

Incremental Release Content

Provide a plan that includes:

1. The number of CSUs to be included in each incremental build/release, and
2. The completion date for each incremental build/release.

For the current reporting period, provide:

1. The number of CSUs that were added to or deleted from each incremental build/release, and
2. Any changes in completion dates for each incremental build/release.

APPENDIX C

Enhanced DID Backup Sheet

Possible uses:

DI-A-7089 Conference Minutes
DI-MGMT-80227/T Contractor's Progress,
Status, and Management Report

The contractor shall substitute or add the following to Blocks 3 and 10 of the DID:

Block 3: to provide the USAF with insight into the software development progress, status, and problem areas.

Block 10: replace section 10 with the following:

The Software Management Metrics are a high-level summary of the status of the software development effort. Specific guidelines for their contents are given in the paragraphs below.

In the context of this DID, the term "current reporting period" shall refer to the interval of time between the past (i.e., most recent) and present submission dates. The frequency of submission for the report is provided in the CDRL item for this DID.

The contractor shall provide data for the metrics listed below in both graphic and tabular form. This information shall be updated as necessary for the current reporting period. The metrics data shall be provided in graphics form at the contract level for each contractor/subcontractor. Graphs shall also be prepared at the CSCI level for certain metrics as designated below. Additional graphs shall be provided as required to address high-risk or problem CSCIs. Graphs show 12 months of history and 5 months of projections where appropriate. *Revised plans may be added to the graphs, but previous plans, including the original plan, shall not be removed.*

Software Size

For the current reporting period, provide in

tabular and graphic form for each CSCI and for each source code language an estimate of:

1. Newly developed Source Lines of Code (SLOC);
2. Reused SLOC — existing code used as is;
3. Modified SLOC — existing code requiring change; and
4. Total SLOC — all code (sum of above).

(Note: SLOC is equal to the count of all nonliteral semicolons (;) in each Ada package.)

Software Personnel

Provide a plan for each contractor/subcontractor that includes, for each month of the contract:

1. The planned number of software personnel,
2. The planned number of Ada software personnel, and
3. The planned number of experienced software personnel.

For the current reporting period, provide for each contractor/subcontractor actual counts of:

1. The total number of software personnel,
2. The number of Ada software personnel,
3. The number of experienced personnel, and
4. The number of unplanned personnel losses.

Software personnel include engineering and management personnel directly involved with software system planning, requirements definition, design, coding, test, integration, documentation, configuration management, and quality assurance. Experienced personnel are defined as those with over 5 years of experience, of which 3 years are with systems similar to the one under development.

Software Volatility

For the current reporting period, provide:

1. The current total number of software requirements,
2. The cumulative number of requirements changes,
3. The number of new Software Action Items (SAIs),
4. The cumulative number of open SAIs, and
5. The number of SAIs open 0-30 days, 31-60 days, 61-90 days, and over 90 days.

An SAI is defined as any discrepancy, clarification item, or requirement issue that must be resolved by either the contractor or the Government.

Computer Resource Utilization

For the current reporting period, provide estimated percentage utilizations of:

1. CPU computation power for each CPU in multi-CPU configurations,
2. On-line memory for each CPU, and
3. I/O channel capacity.

Design Complexity

For the current reporting period, provide:

1. The average design complexity of the 10 percent most complex CSUs,
2. The average design complexity of the 10 percent most complex CSCs,

3. The design complexity of each CSCI, and
4. The number of the top 10 percent most complex CSUs whose design complexity is less than 8, 8 to 20, and over 20.

(Note: See the attached for the definition and discussion of design complexities. (This discussion is contained in appendix E of this document.))

Schedule Progress

Initially provide the number of months in the program schedule.

For the current reporting period, provide:

1. The budgeted cost of work performed (BCWP) for software,
2. The budgeted cost of work scheduled (BCWS) for software, and
3. The number of months in the program schedule (if revised).

Design Progress

Provide a plan that includes, for each month of the contract:

1. The number of System/Segment Design Document (SSDD) software requirements to be documented in Software Requirement Specifications (SRSs) each month,
2. The number of SRS requirements to be allocated and documented as CSCs in Software Design Documents (SDDs) each month, and
3. The number of CSC requirements to be allocated and documented as CSUs in SDDs each month.

For the current reporting period, provide:

1. The number of SSDD software requirements completely documented in SRSs,
2. The number of SRS requirements completely documented as CSCs in SDDs, and
3. The number of CSC requirements completely documented as CSUs in SDDs.

CSU Development Progress

For each CSCI, provide a plan in both graphic and tabular form that includes, for each month of the contract:

1. The number of CSUs to be designed,
2. The number of CSUs to be tested, and
3. The number of CSUs to be integrated.

For the current reporting period, provide for each CSCI in both graphic and tabular form actual counts of:

1. The number of CSUs designed,
2. The number of CSUs tested, and
3. The number of CSUs integrated.

Testing Progress

Provide a test plan that includes, for each month of the contract:

1. The number of CSC tests,
2. The number of CSCI tests, and
3. The number of system tests scheduled to be performed.

For the current reporting period, provide:

1. The cumulative number of CSC, CSCI, and system tests passed;
2. The number of new software problem reports (SPRs);

3. The cumulative number of open SPRs;
4. SPR density, i.e., the cumulative number of SPRs per 1000 SLOC;
5. The number of SPRs that have been open 0-30 days, 31-60 days, 61-90 days, and over 90 days; and
6. The number of CSUs whose SPR density is 0-10, 11-20, 21-30, and over 30.

Incremental Release Content

Provide a plan that includes:

1. The number of CSUs to be included in each incremental build/release,
2. The number of requirements to be implemented in each incremental build/release, and
3. The completion date for each incremental build/release.

For the current reporting period, provide:

1. The number of CSUs that were added to or deleted from each incremental build/release,
2. The number of requirements that were added to or deleted from each incremental build/release, and
3. Any changes in completion dates for each incremental build/release.

APPENDIX D

Glossary

Special Terms

SLOC

Source Lines of Code — includes all source statements created by project personnel and processed into machine code. It excludes comments and unmodified utility software. It includes job control language, format statements, and data declarations. It also includes newly developed support software.

New SLOC

SLOC newly developed, that is, not copied from another source.

Reused SLOC

SLOC copied from another source, then reused without change.

Modified SLOC

SLOC copied from another source and modified.

Reporting Period

A symbol for denoting the intervals for which the Software Management 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, Assembly, 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.

Acronyms

AI	Artificial Intelligence
BCWP	Budgeted Cost of Work Performed
BCWS	Budgeted Cost of Work Scheduled
CDR	Critical Design Review
CDRL	Contract Data Requirements List
COTS	Commercial Off-the-Shelf
CPU	Central Processing Unit
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CSU	Computer Software Unit
DBMS	Database Management System
DID	Data Item Description
IFPP	Instructions for Proposal Preparation
IRC	Incremental Release Content
M	Months
MCCR	Mission Critical Computer Resource
MOTS	Modified Off-the-Shelf
PCA	Physical Configuration Audit

PDL	Program Design Language
PDR	Preliminary Design Review
PMR	Program Management Review
QA	Quality Assurance
SAI	Software Action Item
SDD	Software Design Document
SDR	System Design Review
SM	Staff Months
SOW	Statement of Work
SPO	System Program Office
SPR	Software Problem Report
SRR	System Requirements Review
SRS	Software Requirements Specification
SSDD	System/Segment Design Document
SSR	Software Specification Review
TIM	Technical Interchange Meeting
TRR	Test Readiness Review
V&V	Validation & Verification
WBS	Work Breakdown Structure

APPENDIX E

Design Complexity Definitions (See References 2 and 4)

Cyclomatic Complexity

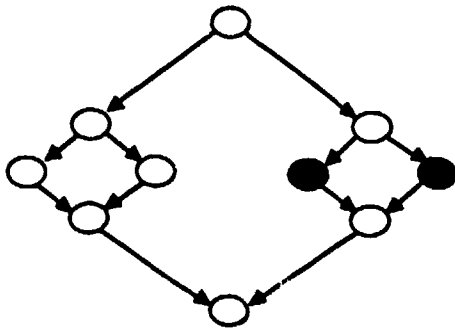
The number of linearly independent paths in a CSU that, when taken in combination will generate every possible path. Generally represented as:

$$C(U) = E - N + 2 \text{ where :}$$

E is the number of connections between nodes, and

N is the number of nodes, e.g.:

$$C(U) = 12 - 10 + 2 = 4$$



In the above example, the complexity is calculated to be 4, indicating that there are four basis paths that will generate every possible path:

A-B-D-H-J

A-C-G-I-J

A-B-E-H-J

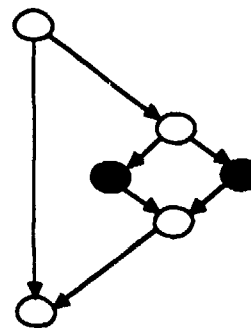
A-C-F-I-J

Basis paths for testing can also be derived from the design complexity calculations in the following paragraphs.

CSU Design Complexity

The cyclomatic complexity of a CSU reduced to include only logic that interfaces with other CSUs and designated as DC(U). A black node in the diagram represents an interface with another CSU.

$$DC(U) = 7 - 6 + 2 = 3$$



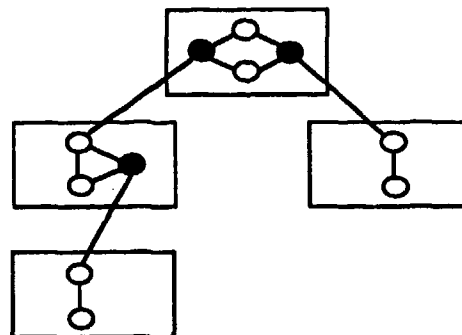
CSC Design Complexity

The number of basis paths or subtrees that, when taken in linear combination, will generate the entire set of subtrees for a CSC. It is equal to the sum of CSU design complexities minus the number of CSUs plus one.

$$DC(CSC) = DC(U_j) - X + 1, \text{ where}$$

DC(U_j) is the design complexity of CSU_j and X is the number of CSUs.

$$DC(CSC) = 6 - 4 + 1 = 3$$



CSCI Design Complexity

The number of basis paths or subtrees that, when taken in linear combination, will generate the entire set of subtrees for a CSCI. It is equal to the sum of CSC design complexities minus the number of CSCs plus one.

$$DC(CSCI) = DC(CSCj) - C + 1,$$

where

$DC(CSCj)$ is the design complexity of $CSCj$ and C is the number of CSCs.