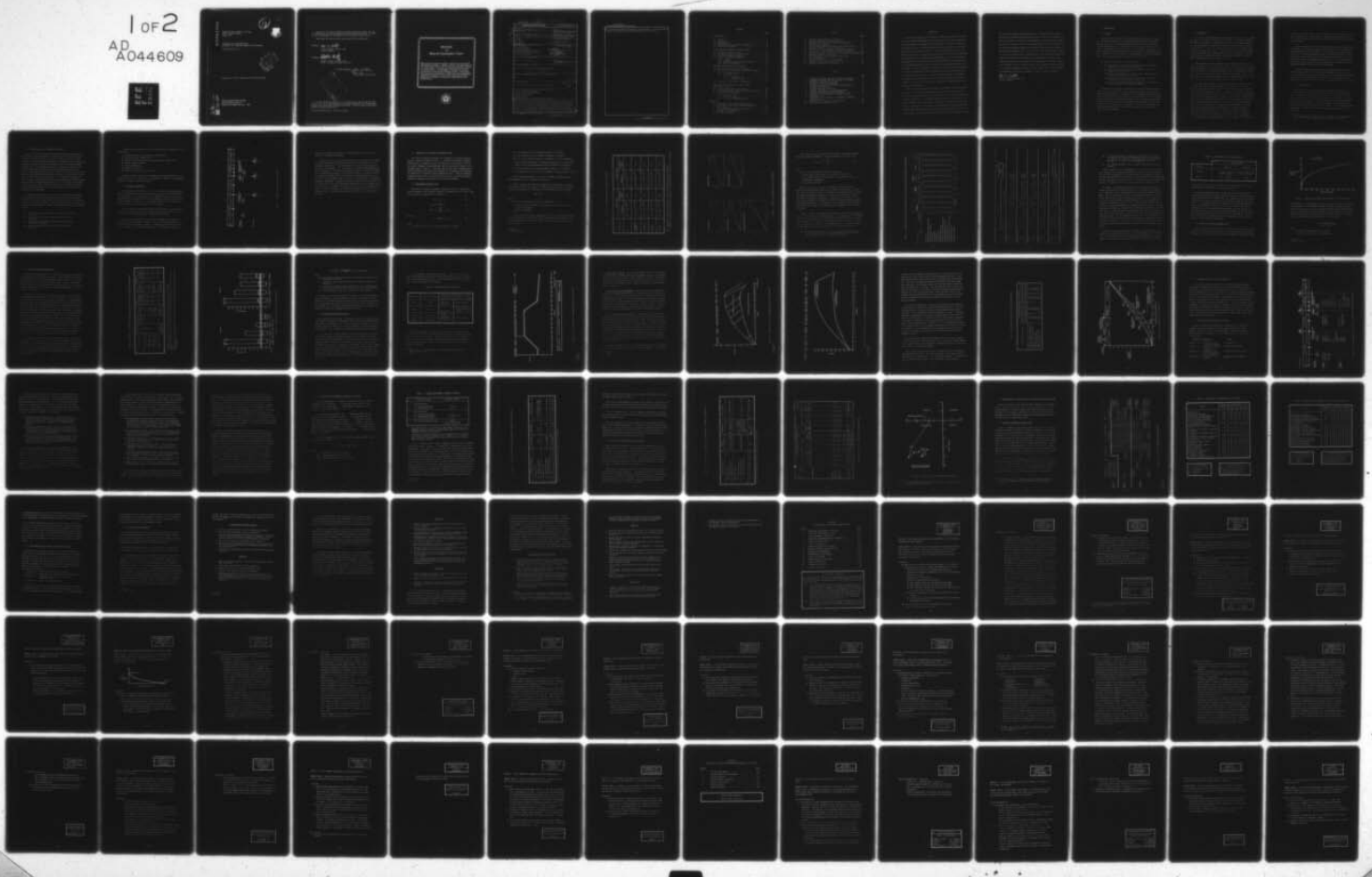
AD-A044 609 DOTY ASSOCIATES INC ROCKVILLE MD F/G 9/2
SOFTWARE COST ESTIMATION STUDY. VOLUME II. GUIDELINES FOR IMPRO--ETC(U)
AUG 77 D L DOTY, P J NELSON, K R STEWART F30602-76-C-0182
UNCLASSIFIED TR-151-VOL-2 RADC-TR-77-220-VOL-2 NL

1 OF 2
AD A044609



(13)

RADC-TR-77-220, Volume II (of two)
Final Technical Report
August 1977

SOFTWARE COST ESTIMATION STUDY
Guidelines for Improved Software Cost Estimating

Doty Associates, Inc.

Approved for public release; distribution unlimited.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

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

This report has been reviewed and is approved for publication.

APPROVED: Alan N. Sukert

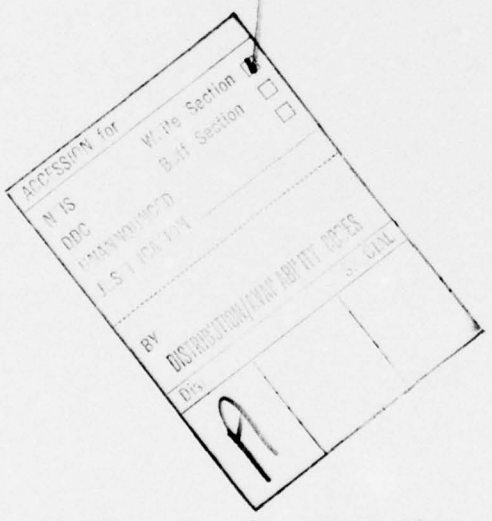
ALAN N. SUKERT, Captain, USAF
Project Engineer

APPROVED: Robert D. Krutz

ROBERT D. KRUTZ, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office



If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (DAP) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-77-220, Volume II (of two)	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): SOFTWARE COST ESTIMATION STUDY. Volume II. Guidelines for Improved Software Cost Estimating.	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, 23 Feb 76 - 23 Feb 77
	6. PERFORMING ORG. REPORT NUMBER: Technical Report No. 151
7. AUTHOR(s): D.L. Doty, P.J. Nelson, K.R. Stewart	8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0182
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Doty Associates, Inc., 416 Hungerford Drive, Rockville MD 20850	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 55811404
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIS), Griffiss AFB NY 13441	12. REPORT DATE: August 1977
	13. NUMBER OF PAGES: 145
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Captain Alan N. Sukert (ISIS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Acquisition Management
Software Cost and Schedule Control
Software Cost Estimating
Software Scheduling
Software Sizing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report contains guidelines for developing estimates of computer software cost. Consideration is first given to the initial program estimate which is often made with a paucity of supportive data. Adjustments are presented for modifying the estimate given the availibility of additional data. Procedures are presented for assessing the affordability of the resulting estimates. Emphasis is placed on developing a conservative but reasonable best estimate for purposes of program budgeting. Separate consideration is given to steps that

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD A06 593

should be taken to bring the program in at or below budget. Frequently recurring problems are summarized in their time-phased order of occurrence.

CONTENTS

FIGURES

TABLES

EVALUATION

The increased importance of software for military applications, coupled with the increased expenditures by both the military and civilian communities for the development of software, has brought about an increased awareness of the present high cost of software and the consistent inability to accurately predict the cost of software projects. This need for producing lower cost software and for more accurately estimating software costs has been expressed in such documents as the Findings and Recommendations of the Joint Logistics Commanders Software Reliability Work Group (Nov 1975) and the Summary Notes of a Government/Industry Software Sizing and Costing Workshop (Nov 1974) (ESD-TR-76-166), as well as in numerous Government and industry sponsored symposia. As a result, several efforts have been initiated to develop better methods for estimating software costs. However, these efforts have not adequately considered the basic underlying factors that affect software sizing and cost estimates, and have not, in most cases, considered non-linear software cost estimating relationships.

This effort was initiated in response to the need to better understand and control those factors that adversely affect software sizing and cost estimates, and fits into the goals of RADC TPO No. 5, Software Cost Reduction (formerly RADC TPO No. 11, Software Sciences Technology), in particular the area of Software Quality (Modeling). The report concentrates on the presentation of guidelines for software cost estimation based upon developed methodologies for controlling over forty factors that have been shown to have an adverse impact on the accuracy of software sizing and cost estimates, in both

the software developer and purchaser domains. The importance of having guidelines for minimizing these adverse factors is that it will enable software cost analysts, as well as software managers, to more accurately predict the costs of software projects, by recognizing those factors that have to be considered when making software cost estimates during the various phases of the software development cycle. This, in turn, will enable software managers to better control the costs of software projects and thus greatly reduce the potential for severe cost overruns that presently exists. Finally, the guidelines proposed in this report will provide methods that future software cost estimators can use to obtain accurate cost estimates during each phase of a software development project, which will greatly aid in the preparation of independent software cost estimates for use in project evaluation.

Alan N. Sukert

ALAN N. SUKERT, Captain, USAF
Project Engineer

1. INTRODUCTION

1.1 Purpose

The purpose of this guide is to provide managers and technical personnel, through the integration of time-phased analytical models and management techniques, with a formalized methodology for improved estimates of software development costs. This guide can be used by all persons interested in estimating and controlling the costs of software development. It does not require specialized capabilities or experience in software development. Therefore, it can be used by program management, software development, and/or fiscal planning personnel to guide their activities.

The purpose of this guide is achieved through:

- Presentation of a model for estimating contractor software development costs by type of application;
- Identification of factors having a significant effect on software development costs;
- Provision of guidelines for mitigating the impact of factors having an adverse effect on costs;
- Specification of a model for estimating total (including government) software development costs; and,
- Discussion of program management guidance for government personnel responsible for software development.

The data used to support the development of the cost estimating models presented in this guide do not reflect modern programming methods nor advanced computer technology. Consequently, cost estimates derived from these models should be used to guide rather than to effect decisions. The methods presented herein are considered improvements over those proposed heretofore; however, the relative results are considered more meaningful than the absolute values of the estimates. Nevertheless, in lieu of similar models derived from more recent data, the enclosed procedure can be considered a viable tool for management.

1.2 Background

The field of software management and engineering is still in its infancy, especially as it relates to deriving cost estimates of software development. The field has evolved to the state where the cost of a software package is generally developed by estimating the number of delivered source or object instructions (i.e., size) and multiplying the size by a cost factor based on average personnel productivity. The Air Force, other DoD and government agencies, and commercial organizations have found this method to be inadequate because this simplistic approach has resulted in large cost overruns in several software development projects.

Size estimates have been observed to be erroneous in many cases by a factor exceeding 3, and it is common to have a productivity factor that has a standard deviation 2.5 times the expected value. With such large variances associated with the two factors most commonly used in software cost estimation, it is not surprising that large software cost overruns occur. Of the two factors, size is the more important since a misestimation in this parameter can have an impact on hardware as well as software costs.

Increased sophistication of software applications over the past ten years has made these erroneous estimates more significant in terms of absolute costs and the percent impact on total system cost. The erroneous estimates can be caused by any one or a combination of numerous factors. Among the most critical factors are changes in the operational requirements, which affect the functional specifications of the software. However, even when the specifications have been fixed, it has been difficult to project the resource requirements accurately. The primary resource--manpower--varies widely in productivity and quality, and is affected in a complex manner by the multi-dimensional environment in which the software is developed. Secondary resources such as machine time and publications support are frequently unavailable at appropriate times. In addition, information with which to develop estimates of resource requirements, such as program size, program language, and type computer, is not always

available on a timely basis. And, if these items are described, the system can be aggregates of so many elements, organizational interactions, logistical considerations, etc., that it is difficult to assess the scope of the work accurately.

During the past several years, extensive work has been performed in the development of techniques and guides for the prediction of software costs and management of software programs. The cost models evolving from these studies have usually been demonstrated to be inaccurate estimators for the reasons discussed previously, erroneous estimation of the size of the software package and/or of programmer productivity. In addition, management guides and control mechanisms have not been properly implemented to ensure adequate management control of software development.

To overcome these deficiencies, more accurate estimators have been derived[1] and integrated into a time-phased management structure to assist in the derivation of more accurate estimators of software development costs. Since effective program management and accurate cost estimation are two dimensions of cost control, both of these areas are addressed in this guide.

1.3 Applicability

This guide is relevant to all Air Force activities responsible for acquiring software as part of major defense systems, such as command and control, managed in accordance with the AFR 800 series, and software as part of automatic data processing (ADP) systems, being managed in accordance with the AFR 300 series. The guide applies to all software developments regardless of whether or not the program is monitored by the DoD Defense System Acquisition Review Council (DSARC). For non-DSARC programs, appropriate adjustments are noted for modifying the schedule and cost estimates.

---

1. Doty Associates, Inc., RADC-TR-77-220, "Software Cost Estimation Study, Vol I: Study Results, Final Report" dated June 1977 (A042264).

1.4 Related directives and other references

A vast collection of Department of Defense and Air Force directives exist that have varying degrees of relevance to the task of software cost estimating. Perhaps the best compendium of these directives is in the "Software Acquisition Management Guidebook: Regulations, Specifications and Standards" prepared by the Mitre Corporation for the Electronic Systems Division (ESD). Brief summary descriptions regarding the content of the primary directives affecting software acquisitions are presented, as well as a comparative discussion of the AFR 300 and 800 series, which notes that "the two series are...not mutually exclusive", and therefore, managers should be familiar with both AFR series. The guidebook is valuable because it helps relieve the new or relatively inexperienced software development manager of the immediate burdensome task of reviewing a large number of applicable directives. Necessary guidance, or at least the identification of specific directives to be read in response to a problem, can be obtained from the guidebook.

The guidebook cited above is one of the "Software Acquisition Management Guidebook" series being prepared under the direction of Electronic Systems Division (ESD), which should be required reading for all managers involved in software development. Four guidebooks, part of a planned series of sixteen (16) applicable to software development management, are currently available:

- Regulation, Specifications and Standards, ESD-TR-75-91, AD-A016401.
- Contracting for Software Acquisition, ESD-TR-75-365, AD-A02044.
- Monitoring and Reporting Software Development Status, ESD-TR-75-85, AD-A016488.
- Software Documentation Requirements, ESD-TR-76-159, AD-A027051.

In addition, the following seven (7) guidebooks will be available in the near future:

- Statement of Work (SOW) Preparation, ESD-TR-77-16
- Life Cycle Events, ESD-TR-77-22
- Software Development and Maintenance Facilities, ESD-TR-77-130
- Software Quality Assurance
- Software Maintenance
- Software Verification
- Software Validation and Certification.

Further, the RADC "Structured Programming Series" (RADC-TR-74-300), consisting of fifteen (15) volumes plus an addendum to Volume VII, is recommended reading for software development managers.

1.5 Guidebook organization

This guidebook is addressed not only to the software development manager, but also to the software specialists whose expertise is required throughout the development. In initial phases of the development, these idividuals will provide inputs for the Required Operational Capability (ROC) and Initial Budgeting Estimates. Therefore, it is important that they have a background in both the technical and costing aspects of the program.

Figure 1 provides a Software Program Life-Cycle overview. Noted on the overview are four points where it is recommended that cost estimates be prepared. This guide presents methods for making those estimates.

Section 2 presents algorithms for estimating software development resources and time requirements. Techniques are also shown for assessing fiscal feasibility of the proposed program. A framework for converting the resource estimates into time-phased program costs is presented in Section 3,

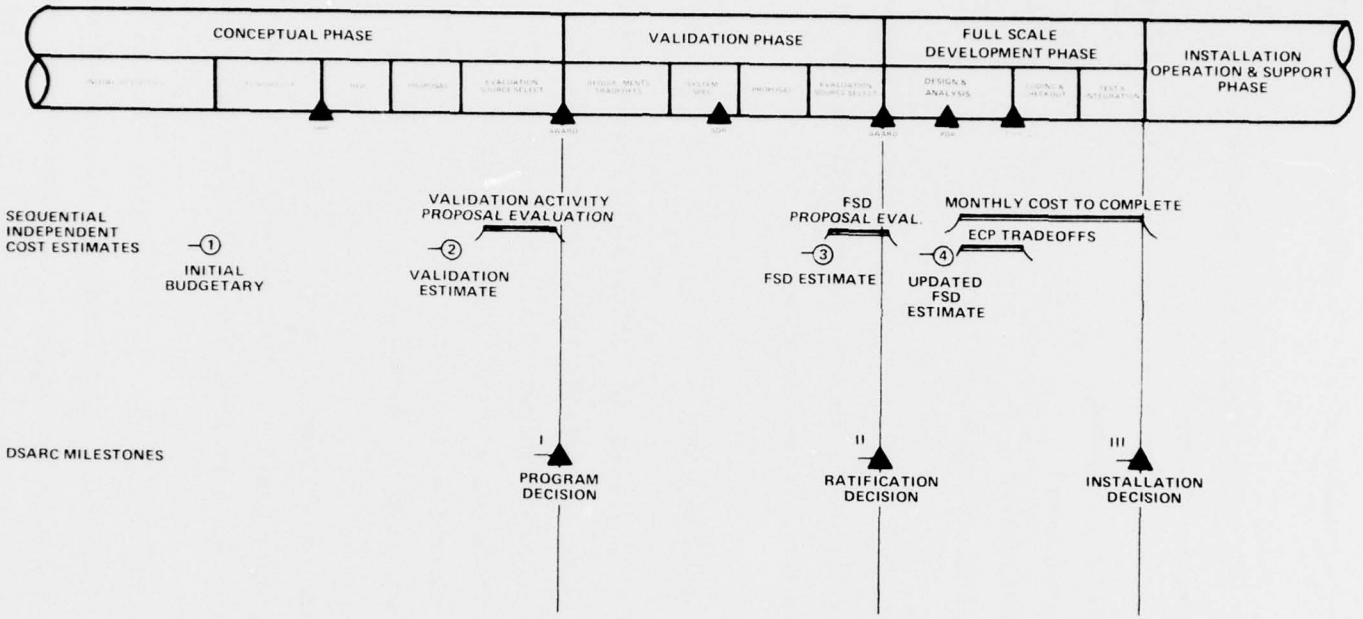


Figure 1. Software Program Life Cycle Overview

and Section 4 discusses pitfalls to be avoided, making use of the cost estimates in managing the program.

Appendices A, B, and C provide discussions of, and controls for, factors which can significantly affect the magnitude of the development costs (or programmer productivity). The factors and their controls have been structured into three domains: (1) Requirements, (2) Architectural/Engineering, and (3) Management. The Requirements Domain (Appendix A) addresses elements of the software requirements; the Architectural/Engineering Domain (Appendix B) encompasses systems design and operation; and the Management Domain (Appendix C) includes elements under the control or responsibility of software development management. Appendix D describes software sizing techniques used in preparation of the Initial Budgetary Estimate, which is often made with a paucity of data. Procedures for updating the initial sizing estimate are presented, given the availability of additional information throughout the development cycle. A glossary of acronyms used in the guide is presented in Appendix E.

2. APPRAISING THE SOFTWARE DEVELOPMENT EFFORT

This section presents algorithms for estimating the resource requirements to software development efforts. A framework for assessing program affordability is presented, giving consideration to the amount of funding that could reasonably be made available and the uncertainty inherent in the estimated level of effort. Guides that show desired rate of expenditures in terms of milestone attainment are presented for the purposes of evaluating the viability of a proposed resource expenditure plan. Since the resources expended are most highly correlated with the size of the software program, methods are described in Appendix D for sizing the software.

2.1 Estimating developer costs

The costs of software development (developer costs) are comprised of primary (manpower) and secondary (computer, documentation, etc.) costs. Thus, the costs can be expressed as the following:

$$C_D = C_p + C_s \tag{1}$$

$$\text{but, } C_p = (MM)\ C_e \tag{2}$$

$$\text{and, } C_s = \sum_{i=1}^{i=n} C_i = k\ C_p \tag{3}$$

Therefore,

$$C_D = (MM)\ C_e\ (1 + k) \tag{4}$$

where

C_D = the total cost of software development, in dollars

C_p is the primary cost of software development, in dollars.

C_s is the secondary cost of software development, in dollars.

MM is the total manpower required, in man-months, for the development.

C_e is the average labor rate, in dollars per man-month, including overhead, general and administration costs, and fees, as appropriate.

C_i is the individual cost of the secondary resource, i, in dollars.

k is a factor which is the ratio of secondary to primary costs (= .075).[2]

Total software development costs, which encompass government support and management costs, are discussed in Section 3.

Table 1 presents algorithms for estimating the total manpower required for analysis, design, code, debug, test and checkout of software as a function of the application. Essentially, the relationships are of the form:

$$MM = a\ I^b \qquad (5)$$

where

MM is the total manpower required, in man-months.

I is the size of the program, in thousands of object words or source lines, as appropriate.

a and b are constants.

The applications encompass command and control, scientific, business and utility packages; the "all" category can be used when the application is for other categories than that shown. Figure 2 presents graphical plots of the ten algorithms.

---

2. Ibid., pg. 73.

TABLE 1. SOFTWARE DEVELOPMENT MANPOWER ESTIMATING ALGORITHMS

Application	Object Code		Source Code	
	Estimator	Standard Error	Estimator	Standard Error
All	$MM = 4.790\ I^{0.991}$	62.2	$MM = 5.258\ I^{1.057}$	50.7
Command and Control	$MM = 4.573\ I^{1.228}$	41.1	$MM = 4.089\ I^{1.263}$	41.1
Scientific	$MM = 4.495\ I^{1.068}$	72.1	$MM = 7.054\ I^{1.019}$	72.1
Business	$MM = 2.895\ I^{0.784}$	12.4	$MM = 4.495\ I^{0.781}$	10.7
Utility	$MM = 12.039\ I^{0.719}$	58.1	$MM = 10.078\ I^{0.811}$	51.7

MM = Man-Months required for analysis, design, code, debug, test and checkout

I = Number of delivered instructions expressed in either Object or Source Code in thousands.

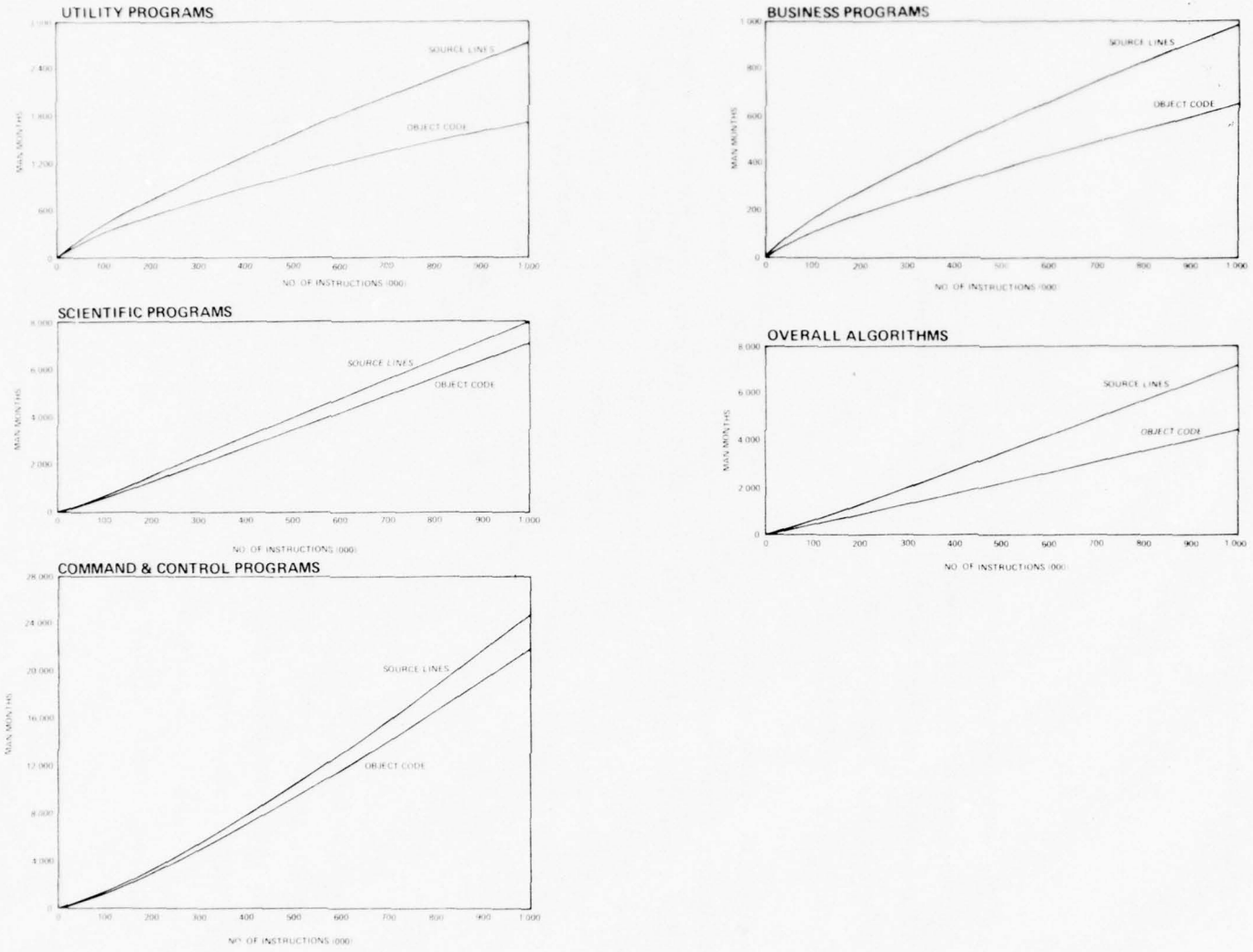


Figure 2. Software Development Resource Estimators

Table 2 also presents algorithms for estimating the manpower requirements for software development. These relationships are of the form:

$$MM = a_1 I^{b_1} \prod_{j=1}^{j=n} f_j \qquad (6)$$

where

MM is the total manpower required, in man-months.

I is the size of the program, in thousands of source lines.

f_j is a constant reflecting the effect of environmental factor, j, on the manpower requirements.

a_1 and b_1 are constrants.

Equation (5) estimates manpower requirements, assuming little is known about the development other than the size of the program. Equation (6), on the other hand, reflects the impact of environmental factors, other than modern programming techniques, considered to have the most significant effect on manpower requirements. Unfortunately, there was no reported data with which to project the effects of modern programming methods on software development. As information becomes available, the estimates derived with these models can be adjusted for anticipated efficiency improvements.

Figure 2 presents the suggested utilization of the estimating relationships. In the analysis and design phase of the development, more information relative to the development environment will become known which will permit the estimators incorporating the environmental factors to be used. These models can be used to support cost estimates derived in subsequent phases of the development; the following guidelines are offered for their use:

- In concept formulation, if the size of the program in object code is known, use the object code estimators. They will give more accurate estimates of manpower requirements.

TABLE 2. SOFTWARE DEVELOPMENT MANPOWER ESTIMATING ALGORITHMS REFLECTING DEVELOPMENT ENVIRONMENT

FACTOR / ESTIMATOR FOR MM:		APPENDIX REFERENCE	APPLICATION									
			ALL		COMMAND & CONTROL		SCIENTIFIC		BUSINESS		UTILITY	
			$2.060\ I^{1.047} \prod_{j=1}^{j=14} f_j$		$0.501\ I^{1.263} \prod_{j=1}^{j=14} f_j$		$2.011\ I^{1.019} \prod_{j=1}^{j=14} f_j$		$3.742\ I^{0.781} \prod_{j=1}^{j=14} f_j$		$1.744\ I^{0.811} \prod_{j=1}^{j=14} f_j$	
		PAGE	YES	NO	YES	NO	YES	NO	YES	NO	YES	NO
SPECIAL DISPLAY	f_1	C-36	1.11	1.00	1.11	1.00	1.11	1.00	1.43	1.00	1.00	1.00
DETAILED DEFINITION OF OPERATIONAL REQUIREMENTS	f_2	A-2	1.00	1.11	1.00	1.54	1.00	2.00	1.00	1.00	1.00	1.00
CHANGE TO OPERATIONAL REQUIREMENTS	f_3	A-5	1.05	1.00	1.05	1.00	1.05	1.00	1.05	1.00	1.05	1.00
REAL TIME OPERATION	f_4	A-8	1.33	1.00	1.33	1.00	1.67	1.00	1.00	1.00	1.43	1.00
CPU MEMORY CONSTRAINT	f_5	B-4	1.43	1.00	1.25	1.00	1.25	1.00	1.00	1.00	1.18	1.00
CPU TIME CONSTRAINT	f_6	B-2	1.33	1.00	1.51	1.00	1.67	1.00	1.00	1.00	2.32	1.00
FIRST S/W DEVELOPED ON CPU	f_7	C-35	1.92	1.00	1.92	1.00	1.92	1.00	1.92	1.00	1.92	1.00
CONCURRENT DEVELOPMENT OF ADP H/W	f_8	C-33	1.82	1.00	1.67	1.00	2.22	1.00	1.33	1.00	1.25	1.00
TIME SHARE, VIS-A-VIS BATCH PROCESSING, IN DEVELOPMENT	f_9	C-11	0.83	1.00	0.83	1.00	0.83	1.00	0.83	1.00	0.83	1.00
DEVELOPER USING COMPUTER AT ANOTHER FACILITY	f_{10}	C-30	1.43	1.00	1.43	1.00	1.43	1.00	1.43	1.00	1.43	1.00
DEVELOPMENT AT OPERATIONAL SITE	f_{11}	C-29	1.39	1.00	1.39	1.00	1.39	1.00	1.39	1.00	1.39	1.00
DEVELOPMENT COMPUTER DIFFERENT THAN TARGET COMPUTER	f_{12}	C-23	1.25	1.00	2.22	1.00	1.11	1.00	1.00	1.00	1.43	1.00
DEVELOPMENT AT MORE THAN ONE SITE	f_{13}	C-31	1.25	1.00	1.25	1.00	1.75	1.00	1.25	1.00	1.21	1.00
PROGRAMMER ACCESS TO COMPUTER	f_{14}	C-11	LIM.: 1.00 UNLIM.: 0.90		1.00 1.00		1.00 0.67		1.00 0.90		1.00 0.67	

Application	Phases of Development: Concept Formulation Phase	Analysis and Design Phase	Subsequent Phases
All Software			
- Object	$MM = 4.790\ I^{0.991}$	$MM = 4.790\ I^{0.991}$	
- Source	$MM = 5.258\ I^{1.047}$	$MM = 5.258\ I^{1.047}$ (for $I \geq 10{,}000$) $MM = 5.258\ I^{1.047}$; (for $I < 10{,}000$)[1] $MM = 2.060\ I^{1.047} \prod_{j=1}^{j=14} f_i$	
Command and Control			
- Object	$MM = 4.573\ I^{1.228}$	$MM = 4.573\ I^{1.228}$	
- Source	$MM = 4.089\ I^{1.263}$	$MM = 4.089\ I^{1.263}$ (for $I \geq 10{,}000$) $MM = 4.089\ I^{1.263}$; (for $I < 10{,}000$)[1] $MM = 0.501\ I^{1.263} \prod_{j=1}^{j=14} f_i$	
Scientific			
- Object	$MM = 4.495\ I^{1.068}$	$MM = 4.495\ I^{1.068}$	
- Source	$MM = 7.054\ I^{1.019}$	$MM = 7.054\ I^{1.019}$ (for $I \geq 10{,}000$) $MM = 7.054\ I^{1.019}$; (for $I < 10{,}000$)[1] $MM = 2.011\ I^{1.019} \prod_{j=1}^{j=14} f_i$	
Business			
- Object	$MM = 2.895\ I^{0.784}$	$MM = 2.895\ I^{0.784}$	
- Source	$MM = 4.495\ I^{0.781}$	$MM = 4.495\ I^{0.781}$ (for $I \geq 10{,}000$) $MM = 4.495\ I^{0.781}$; (for $I < 10{,}000$)[1] $MM = 3.742\ I^{0.781} \prod_{j=1}^{j=14} f_i$	
Utility			
- Object	$MM = 12.039\ I^{0.719}$	$MM = 12.039\ I^{0.719}$	
- Source	$MM = 10.078\ I^{0.811}$	$MM = 10.078\ I^{0.811}$ (for $I \geq 10{,}000$) $MM = 10.078\ I^{0.811}$; (for $I < 10{,}000$)[1] $MM = 1.744\ I^{0.811} \prod_{j=1}^{j=14} f_i$	TIME CONSTRAINED $MM = 10.078\ I^{0.811}$; TIME UNCONSTRAINED $MM = 1.426\ I^{1.243}$

1. Can be used for budgeting for programs of $I \geq 10{,}000$.

Figure 3. Suggested Utilization of Estimating Relationships for Development Manpower

- If accurate estimates of manpower requirements are required in the analysis and design, and subsequent phases of development, use Equation (5) in source code for programs of $I \geq 10{,}000$ and Equation (6) in source code for programs with $I < 10{,}000$.
- For budgetary purposes, use the equation that gives the higher estimate.

Table 3 summarizes these recommendations. In evaluating the relationships, Equation (5), in general, was found to estimate higher for programs with $I < 10{,}000$ and Equation (6) estimated higher for programs of $I \geq 10{,}000$. This perhaps reflected unique facets of the data from which the estimates were derived. Therefore, to be conservative, it is suggested that, for budgeting purposes, the equations that give higher estimates of manpower be used.

In summary, the procedure and models used in estimating the cost of software development are dependent upon what is known about the software (e.g., application, size, type code) and the software development environment. Equation 4 evaluates the cost of software development as a function of the total manpower used in developing the software, the average labor rate, and the ratio between manpower (primary) costs and secondary costs (computer, documentation, etc.). Table 1 presents the algorithms that can be used to estimate the man-months of manpower required to develop software when minimal information is known about the program (application, size, and type code). As more information becomes known about the software development, the algorithms in Table 2 can be used to calculate the estimated total manpower required. The environmental constant, f_j, is the product of the individual factors identified in yes-no assessments of elements of the development environment. If the status of some factors is not known, projections can be made as to their status in order to derive the manpower estimates.

Estimates involving program size in object code should always be derived using the algorithms in Table 1. When the program size is in source code, the algorithms used are also dependent upon program size. For programs of less than 10,000 lines of source code, the algorithms in Table 2

TABLE 3. RECOMMENDED FORM OF RELATIONSHIPS

	SIZE PROGRAMS	
FUNCTION	$I < 10{,}000$	$I \geq 10{,}000$
Budgeting	The form that gives higher estimate.	The form that gives higher estimate.
Planning	$MM = aI^b \prod_{j=1}^{j=n} f_j$	$M = aI^b$

should be used, and for those of equal to or greater than 10,000 lines of source code, the algorithms in Table 1 should be used.

The specific algorithms taken from the tables are also affected by type program (command and control, scientific, business, utility, and all). As noted previously, the algorithms for "all" software should be used when the application cannot be categorized or is different than the categories noted. If a program encompasses a combination of applications and a projection can be made as to the mix (e.g., out of 50,000 source lines of instruction, 10,000 are categorized as business, 10,000 as utility, and 30,000 as scientific), then the appropriate algorithms can be used to estimate the manpower required for those portions of the software. The manpower required for the total program is then determined by summing that required for the individual categories.

2.2 Estimating software development time

Figure 4 presents a graphical plot of an algorithm to estimate the time required for software analysis and design, code and checkout, and test and integration. It should be noted, however, that manpower loading can affect

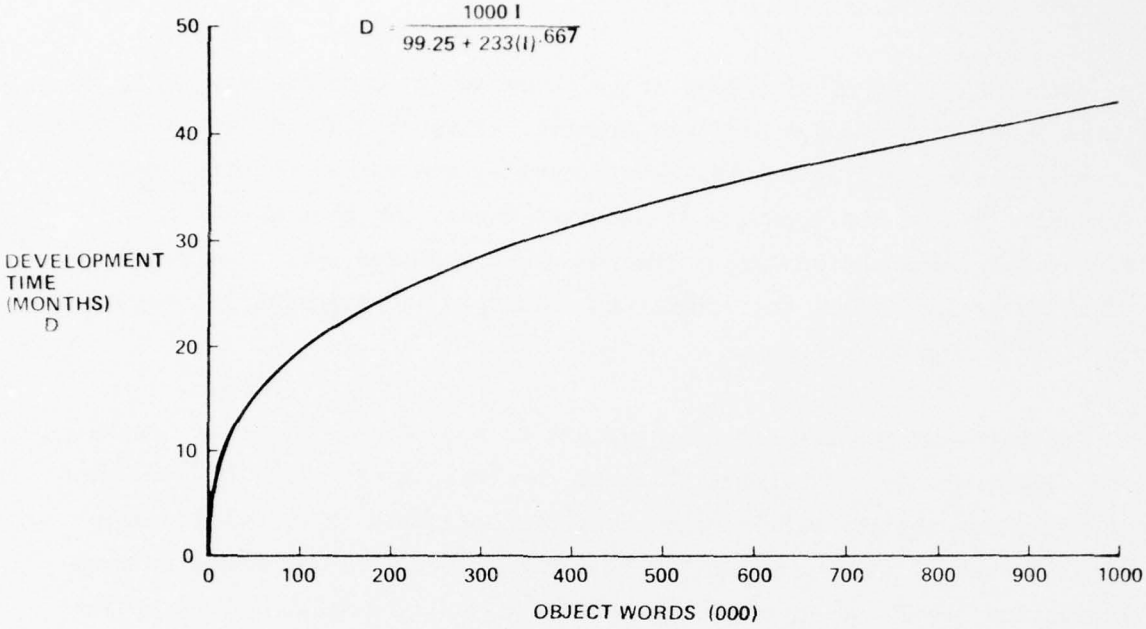


Figure 4. Software Development Time Estimator for Overall Usage

this scheduling. This algorithm, derived by applying regression techniques to a data base of 74 development programs,[3] implies customary manloading. Schedules based on the use of this algorithm should produce reasonable completions which avoid the pitfalls inherent in "crash projects" and the unnecessary time delays resultant from long, drawn out projects. The algorithm for estimating the development time (D) is given as:

$$D = \frac{1000\ I}{99.25 + 233(I)^{.667}} \qquad (7)$$

where

D = Reasonable development time in months.

I = Number of delivered object instructions.

---

3. Ibid., pg. 43.

2.3 Assessing program affordability

During the conceptual phase, it is important to commence examining the fiscal feasibility of the proposed program. This evaluation should be conducted periodically through the development of the software until program affordability, or the adequacy of budgeted funds, has been established. This is determined by analyzing the resources required over a range of programmatic assumptions. The estimators developed in paragraph 2.1 can be used for this purpose.

Estimates of resources can be subject to appreciable error if little or no information is available to guide the estimation. And, the evaluation of affordability with highly erroneous estimates is of questionable value. However, the estimates become progressively more accurate as more information about the software becomes known. For example, having projections of program size (in source or object code), the portion of code to be used in data areas (vis-a-vis executable code), the amount of reusable code, and the language mix (HOL/MOL) of the resultant code can enhance the accuracy of the resource estimates appreciably. This is demonstrated in Table 4 which summarizes the resource estimates for two examples. Both are software programs estimated to consist of 60,000 object instructions. The sequential improvement in man-month accuracy through the availability of more information is very clear. The smaller spread in the absolute values of manpower estimates for Program A is attributed to the fact that the amount of code to be written is less than for Program B (Program A has more code set aside for data areas and has more reusable code).

These examples also illustrate how the potential trends can confirm affordability of a program, or the need for more information or for a design change. The data from Table 4 are plotted in Figure 5. Superimposed on this figure is the level of assumed available funding (in terms of man-months). This value is developed in dialogue with prospective project sponsors. The conversion of dollars to development man-months for the assumed examples is:

TABLE 4. ILLUSTRATIVE EXAMPLES OF PROGRAM FISCAL FEASIBILITY (GENERAL PURPOSE PROGRAM)

Basis of Estimate	Estimator Accuracy[1]		Program A				Program B			
	Sizing Error	Resource Estimator Error	Programmatic Assumptions	Man-months			Programmatic Assumptions	Man-months		
				+1σ	Mean	-1σ		+1σ	Mean	-1σ
1. Total Object Code	200%	62%	60,000	1346	277	57	60,000	1346	277	57
2. Total Object Code minus data areas	100%	62%	40,000 (33% data)	599	185	57	57,000 (5% data)	852	263	81
3. Total Object Code minus data areas or reusable code	75%	62%	25,000 (37.5% reusable code)	329	116	41	54,150 (5% reusable code)	710	250	88
4. Total new Source Code	50% improving to zero at completion	51%	16,667 17% HOL (4:1)[2] 83% MOL (1:1)[3]	227	100	44	49,679 3% HOL (4:1) 97% MOL (1:1)	713	314	139

1. Estimator Accuracy is the standard error expressed as percent of best estimate.
2. High Order Language (HOL) which produces four object instructions per each source instruction written.
3. Machine Oriented Language (MOL) which produces one object instruction for each source instruction written.

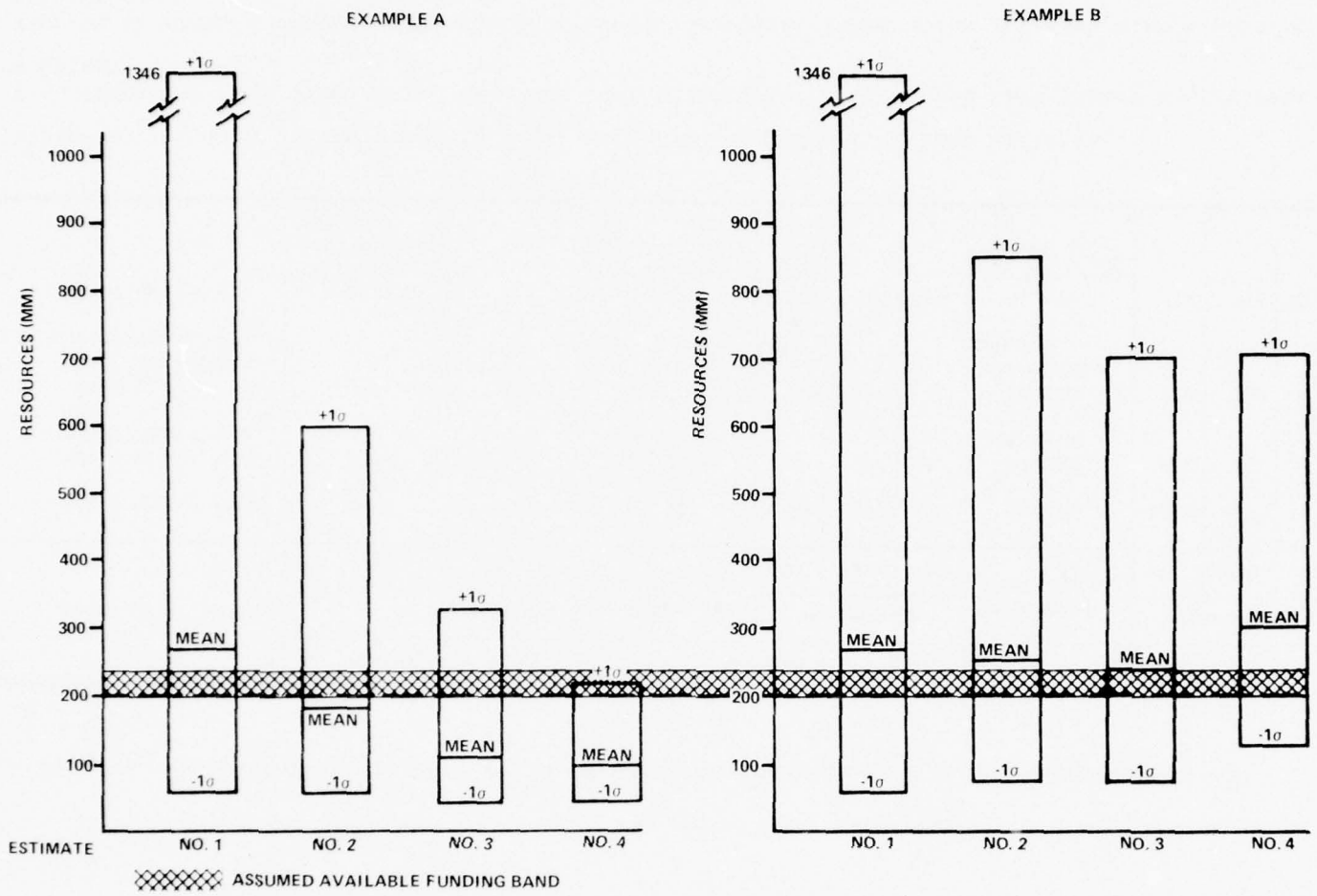


Figure 5. Exemplary Analysis of Fiscal Feasibility

$$MM = \frac{F\ K_2}{K_3} = \frac{\$1,800,000 \times .6}{\$5,000} = 216 \text{ man-months} \qquad (8)$$

where

MM = Full-Scale Development (FSD) effort in man-months required by the development contractor.

F = $1,800,000 total program funding level in dollars assumed to be available.

K_2 = .6 fraction of total program represented by the contractor's FSD effort (see paragraph 3.1 for discussion of other program phases).

K_3 = $5,000 per month, contractor's average cost per man-month.

In Figure 5, it can be observed that Program A is clearly more attainable than Program B, and that Program B should not be pursued in its proposed form. The analysis further provides a basis for examining sensitivity to key programmatic assumptions such as assumed data areas, reusable code, and language mix. The example also demonstrates the need for updating the estimates as additional information becomes available.

2.4 Resource expenditure evaluation

The discussion which follows is intended to assist the software program cost analyst in evaluating a Development Activity's proposed Computer Systems Resource Development Plan. Data are presented that show desired rate of expenditure as a function of milestone attainment for various software project levels (size, complexity, and management implementation structure).

2.4.1 Schedule. Table 5 summarizes the expected completion times for various size software development projects. This is offered as a first check for schedule reasonableness. Should a proposed schedule fall outside the bounds noted, questions should be asked as to whether there is some reasonable basis for it. For example, a development schedule for a program consisting of large data areas and extensive reusable code may have a large total object code count which could be completed in less time than that noted in the table. A second check for reasonableness can be made by comparing the proposed schedule with that which is obtained from using Figure 4 (p. 17).

2.4.2 Management implementation structure. Table 5 also lists the types of implementation structures that are reasonable to apply for various size software development projects. The discussion which follows describes each of these structures.[4] They include:

TABLE 5. SOFTWARE MANAGEMENT STRUCTURES

Schedule, mo.	Total object code	Management implementation scheme	
		Normal pyramid	Structured top-down
12-18	<100K	First level project	Savings unknown
18-24	100-200K	Second level project build-up and bottoms-up integration and test	<40% decrease in level of required effort
24-36	200-600K		Savings unknown
36-48	600-1500K		

First Level Projects. A smaller project that can be accomplished by a single first level group manager who often assigns his personnel as a team for the duration of the project. In many cases, the resource expenditures curve for these projects tend to have less buildup and decline in manpower loading than second level projects. Figure 6 illustrates such a distribution of resources.[5]

4. Aron, Joel D., "Characteristics of Systems Development Life-Cycle", IBM Corporation, 1973.
5. Ibid

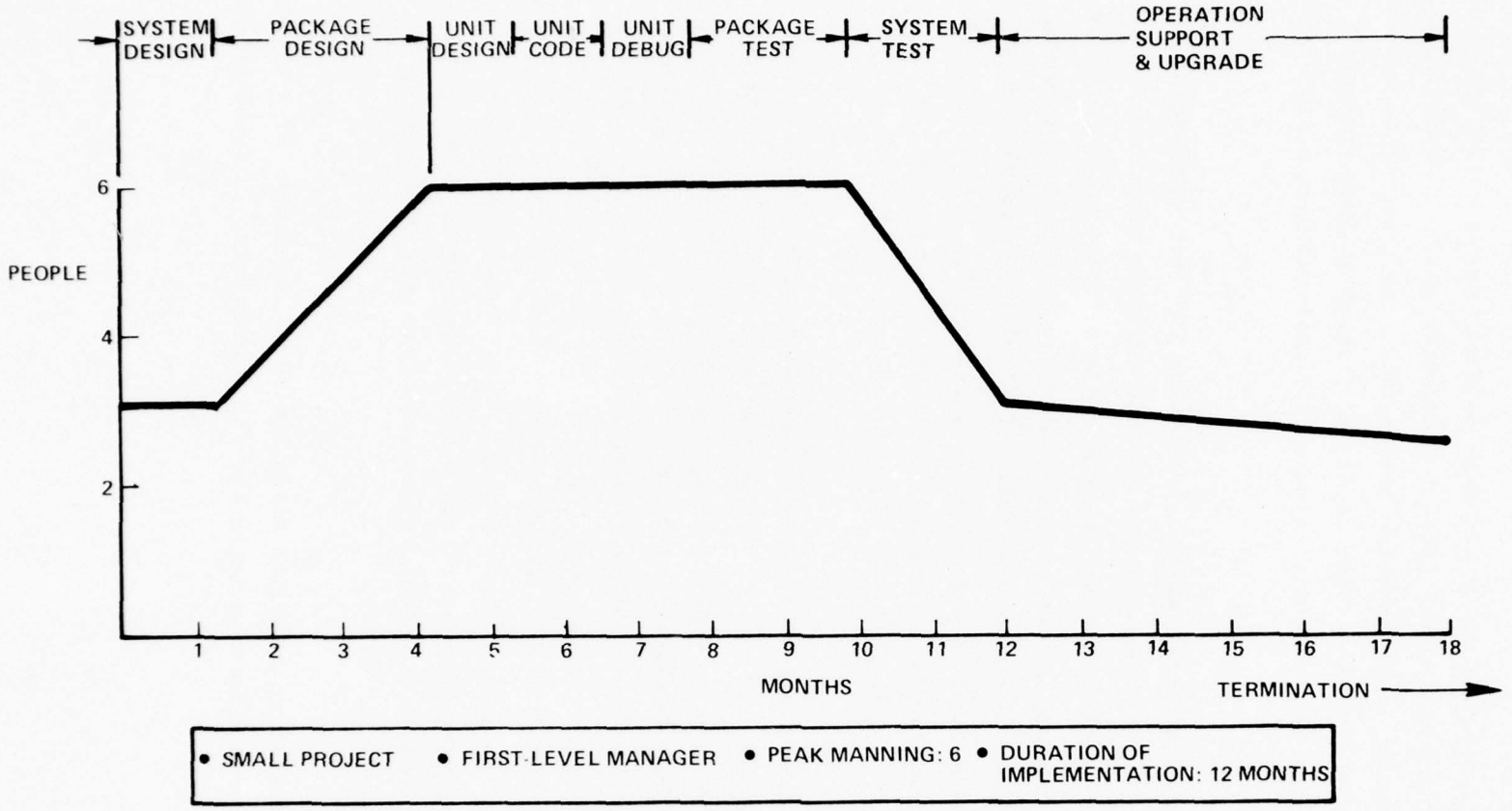


Figure 6. First Level Project Management Resource Distribution[6]

6. Ibid.

Second Level Projects. Large projects managed under a normal pyramid hierarchy require a second level project management group for accomplishing the actual design, coding, system integration, and test. This second level group is supported by a first level group which is approximately equal in size to the second level. Figure 7 illustrates such a distribution of resources.[7]

Structured Top Down Programming. Structured Top Down Programming is a practice involving top down design of a program using structured code. Top down design first involves the design of the software module with the highest level of control in the program, and then proceeds to the design of successively lower level modules until the level is reached at which algorithms are programmed in source code. Structured code is a manner of organizing the code whereby a program has one entry and one exit point, and there is careful indentation of the code to show nesting levels.

These practices have been applied to moderately large projects with considerable success, and savings of up to 40 percent have been reported. Top down design results in decreased costs because integration and test of the modules occur as they are developed. Also, if desired, deliveries on individual modules are possible. Since problems are resolved as they occur, special integration and test teams are not needed, and the complexity that can evolve in bottoms-up development is avoided. Test cases are inherent in the design thereby decreasing required support. Documentation is simplified because it can be easily generated in parallel with programming rather than being aggregated from unit descriptions during the test phase.

Figure 8 illustrates a typical distribution of resources for a moderately large project using structured top down programming.[8] Superimposed

---

7. Ibid.

8. Ibid.

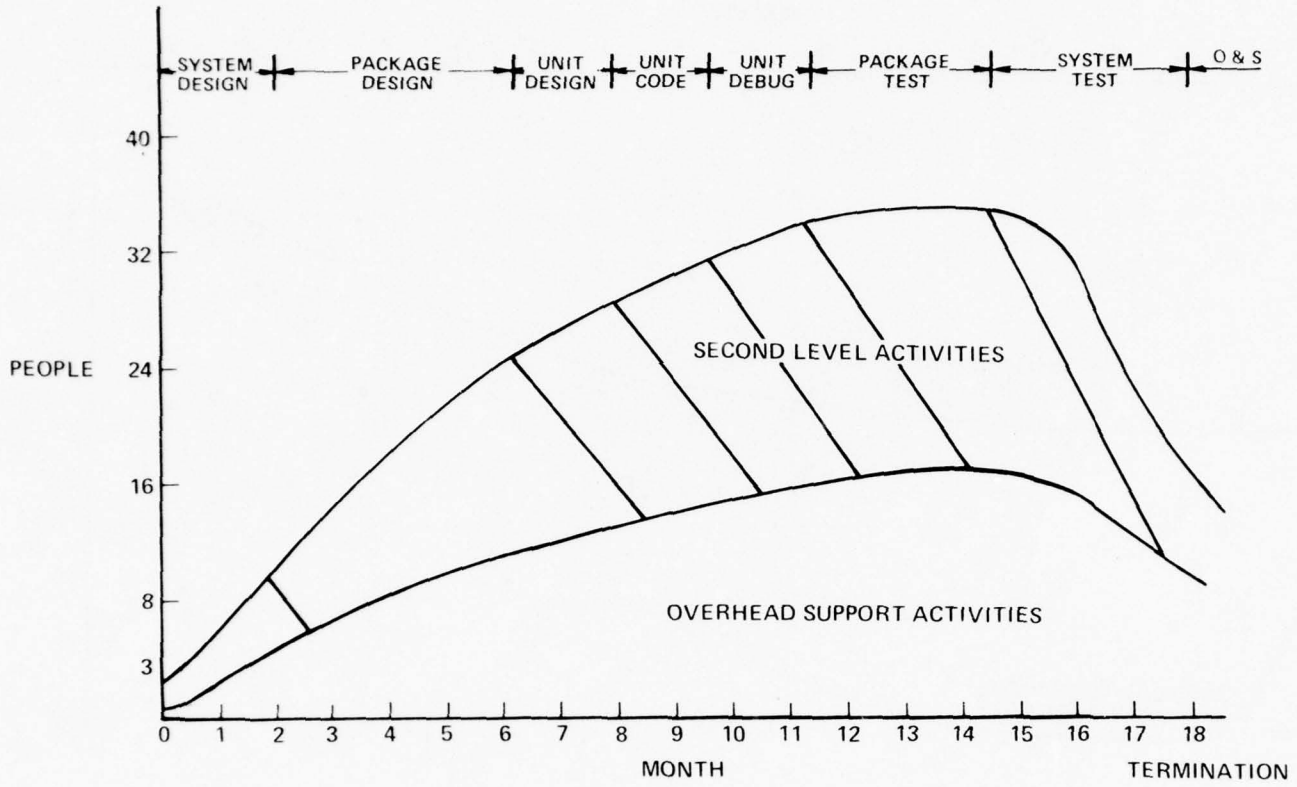


Figure 7. Second Level Project Management Resource Distribution[9]

9. Ibid.

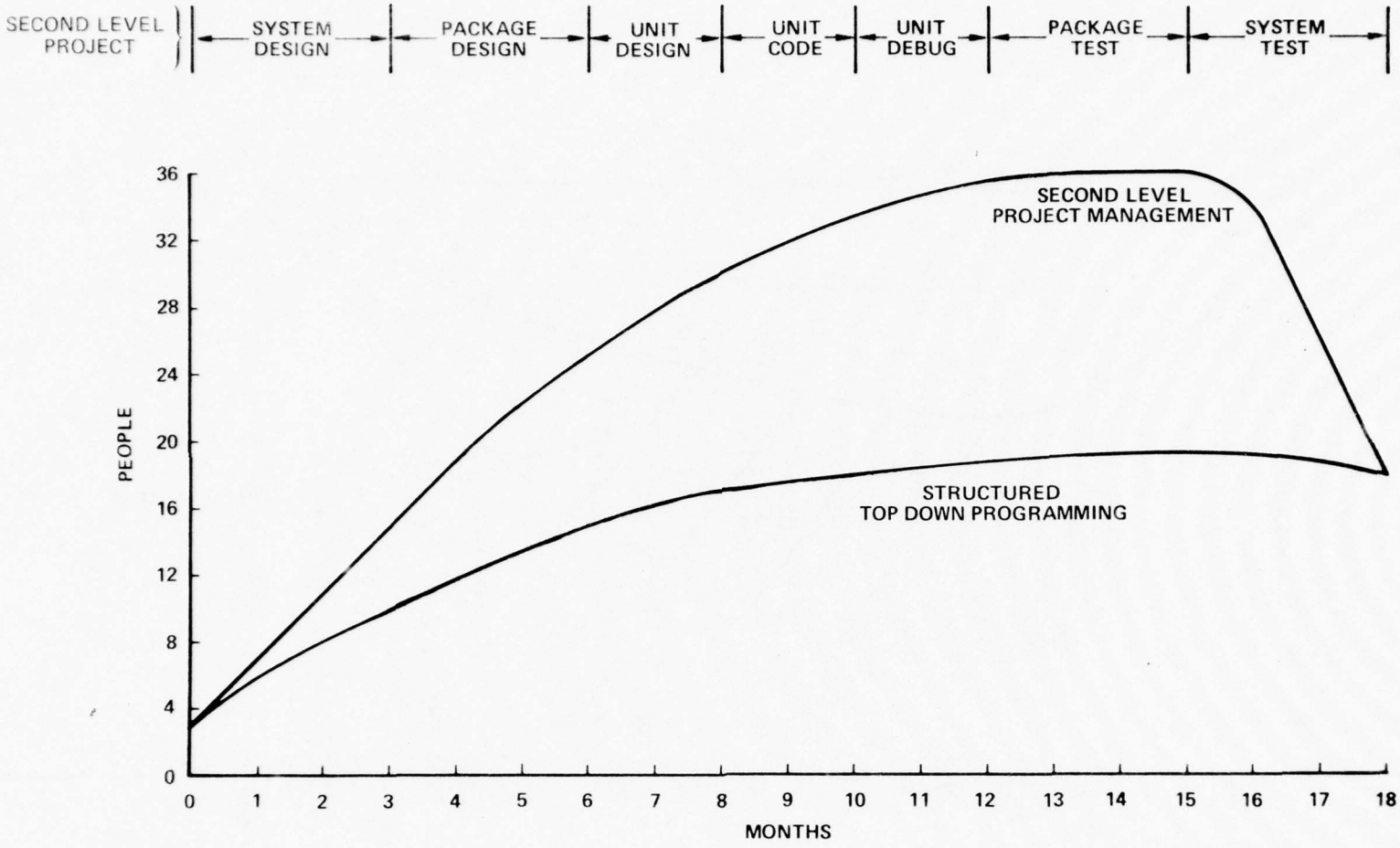


Figure 8. Structured Top Down Project Management Resource Distribution[10]

10. Ibid.

on Figure 8 is the magnitude and distribution of effort required for the same project, using a Second Level Project Management Structure. Note that the program milestones apply only to the Second Level Project Structure. While the milestones are similar in Structured Top Down Programming, they apply to individual modules which are distributed throughout the development cycle. Whether or not such deliveries are sought by the customer, it behooves the customer to conduct Preliminary Design Review (PDR) and Critical Design Review (CDR) at both the CPCI, which is the Computer Program Configuration Item, and the module level, which consists of distinct program logic packages as identified by the user, analyst and/or programmer.

2.4.3 Rate of expenditure. Table 6 lists reasonable distributions of resources as a function of various development milestones. These data are presented for a pyramidal hierarchy type management implementation structure. These same data are shown graphically in Figure 9. Proposed schedules and expenditure plans can be compared against these norms for assessing reasonableness. Departures from the norms should be questioned. For example, an expenditure rate higher than the norm should result in earlier than normal milestone attainment. If not, determine the contractor's understanding of the performance specifications and design. Perhaps the developer has an inefficient use of resources due to the fact that people are available for work prior to the time their tasks have been adequately defined.

Expenditure rates lower than norm imply that the rate must increase quickly at some point. Where does this occur? Are these increased expenditures compatible with the contractor's proposed milestones? If not, suspect a lack of problem understanding.

Figure 9 also shows expected rates of expenditures for Structured Top Down management structure. This rate is similar to that experienced for the smaller, First Level type project management structure. As noted previously, the phases do not apply to the Structured Top Down approach.

TABLE 6: DESIRABLE DISTRIBUTION OF DEVELOPMENT EFFORT

Development Milestones	Desirable Distribution of Effort			
	First Level Project		Second Level Project	
	Schedule	Expenditure	Schedule	Expenditure
Complete System Design (PDR)	10%	5%	10%	1%
Complete Package Design	35%	27%	35%	13%
Complete Unit Design (CDR)	44%	36%	42%	19%
Complete Unit Code	54%	49%	50%	28%
Complete Unit Debug	64%	59%	57%	38%
Complete Package Test	81%	78%	80%	73%
Complete System Test	100%	100%	100%	100%

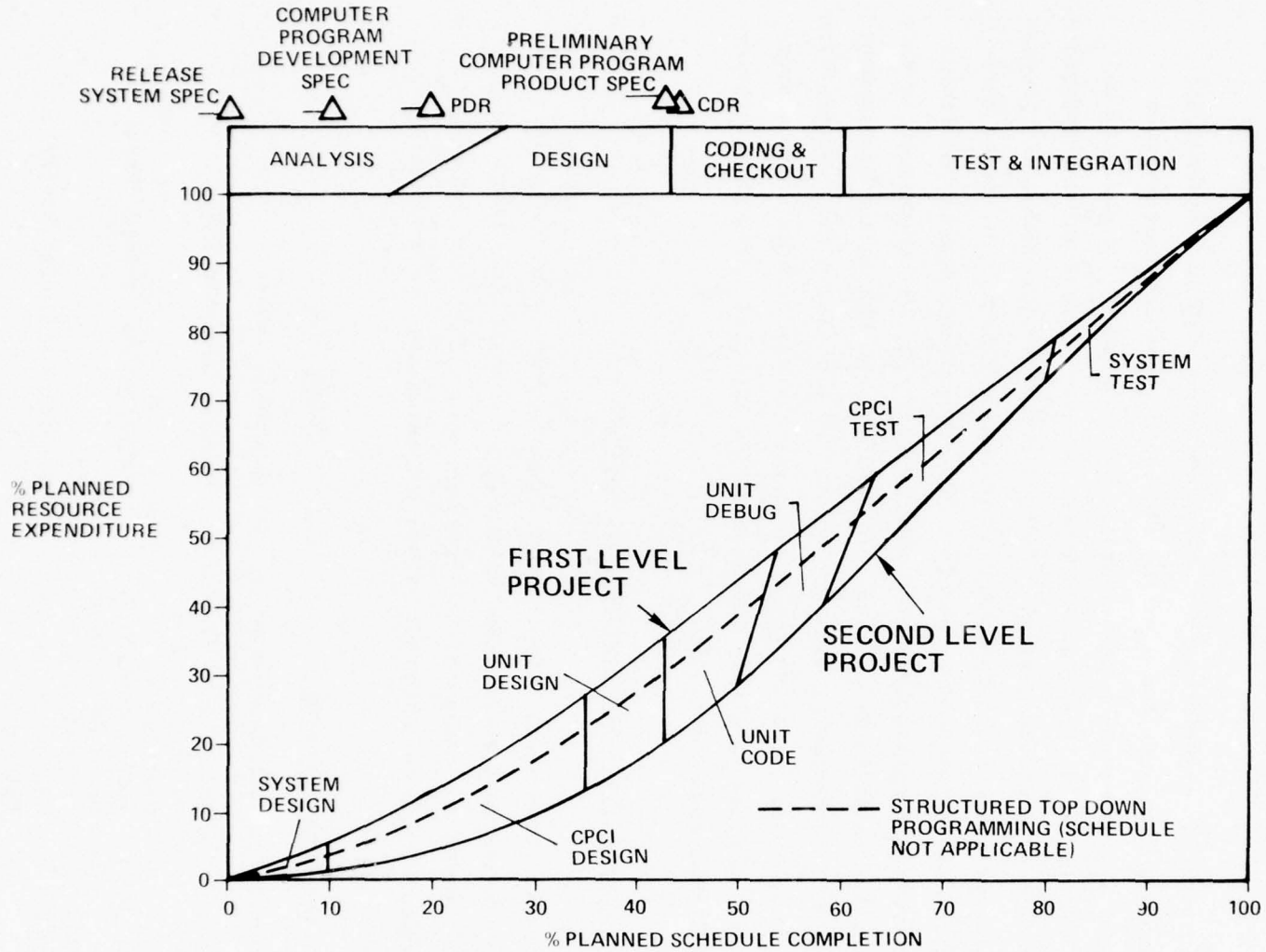


Figure 9. Desirable Resource Distribution

3. SOFTWARE PROGRAM COSTING AND MONITORING

This section defines the role of the software program cost analyst and presents a framework for constructing the program independent cost estimates. Key areas where the expertise and counsel of the analysts will play an important role are also identified. The costing framework integrates the resource and schedule estimators into a procedure for estimating program costs. The resulting cost estimate is time-phased, satisfies the requirements of DoD Directive 5000.1, and includes provisions for both government in-house costs and contractor development costs. As new information is made available, the estimate is updated. In addition, points in the development cycle where the independent estimate is updated are also noted. Acquisition events and analyses requiring these estimates are also listed. The discussion of cost monitoring covers the Request for Proposal (RFP) phase, system implementation, and monitoring.

3.1 Role of software program cost analyst

Figure 10 shows areas throughout the development cycle where the software program cost analyst can make a major contribution to the program. During the course of the development at least four independent estimates will be prepared. These estimates are, in their order of occurrence:

Software Cost Estimate	When
Estimate No. 1 - Initial Program Budgetary Estimate	Conceptual Phase
Estimate No. 2 - Independent Program Validation Cost Estimate	Program Decision Phase
Estimate No. 3 - Independent Full-Scale Development (FSD) Cost Estimate	Ratification Decision Phase
Estimate No. 4 - Update of FSD Cost Estimate	Preliminary Design Review (PDR)

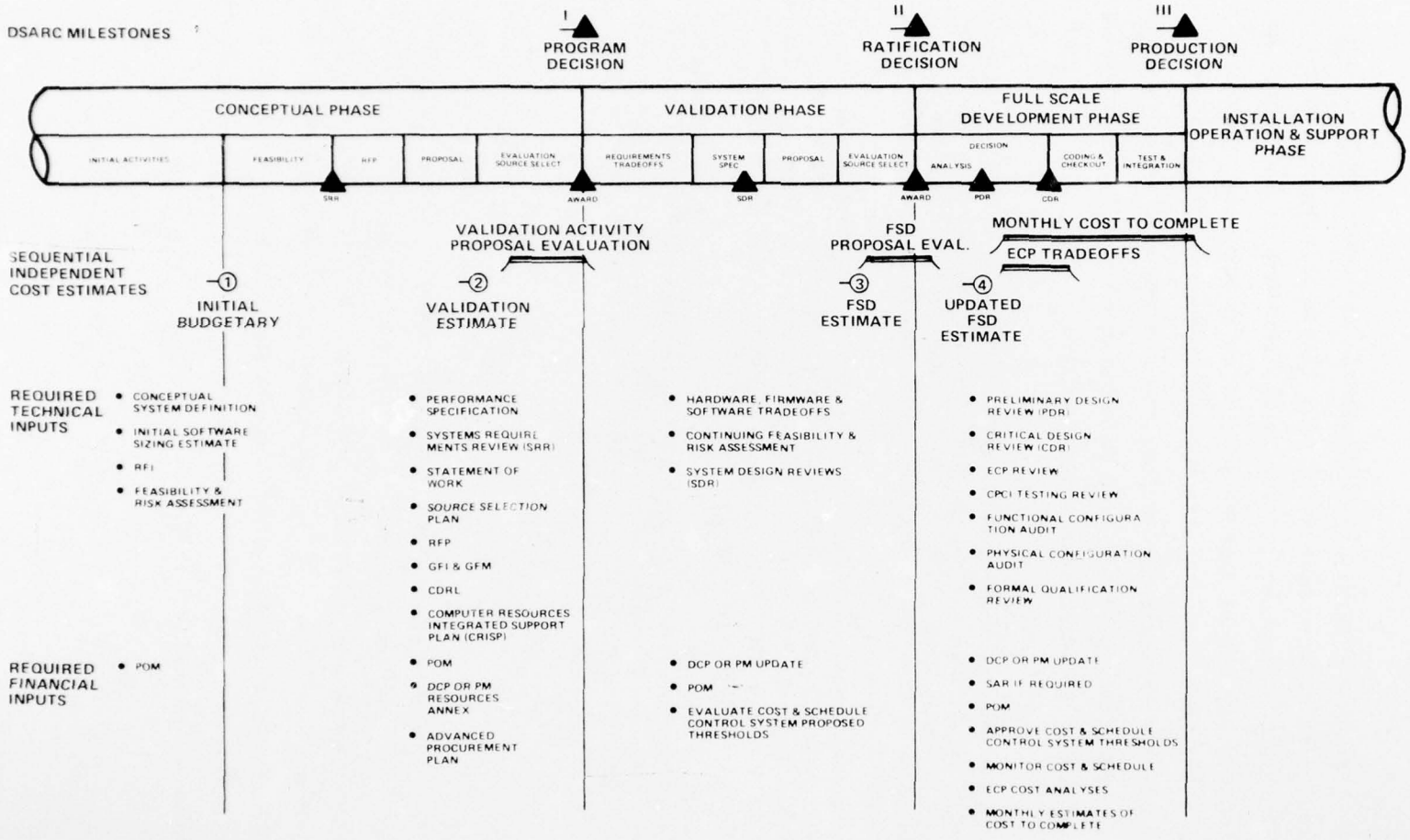


Figure 10. Key Programmatic Areas Requiring Software Cost Analysts Expertise

The initial budgetary estimate is perhaps the most important and most difficult estimate to make since very little information is available at this early conceptual stage. If the estimate is too high, the program probably may not be approved, or if it is approved, Parkinson's Second law, "Expenditures will rise to meet income," will surely come into play. If the estimate is too low, the program will incur overruns. This initial budgetary estimate is a life-cycle cost (LCC) estimate which is used in several key documents. These include:

- Program Objectives Memoranda (POM) - The POM is used by the services to obtain program approval and inclusion in the Five Year Defense Plan (FYDP). The initial budgetary estimate appears in the POM as a time-phased cost by appropriation category.
- Advanced Procurement Plan (APP) - The APP is used by the Development Command to seek procurement approval. Costs are shown apportioned to respective development activities. Also, like the POM, the cost estimate appears in the form of time-phased costs by appropriation category.
- Resources Annex - The resources annex appears in both Decision Coordinating Papers (DCPs) and Program Memoranda (PMs). Like the POM, the resources annex presents time-phased costs by appropriation category. The resources annex is updated each time the DCP or PM is updated.

During the Conceptual Phase it may become desirable to obtain an industrial input. This can either be done informally through existing working relationships, or if done formally, through a Request for Information (RFI). Dollar wise, these inputs are not usually costly. Most of the government's conceptual studies are accomplished in-house. This may change over time with the advent of Office of Management and Budget (OMB) Circular A-109. This regulation is designed to solicit industrial involvement back to the point of evolving technical approaches given statements of mission needs.

Once it appears there is going to be a program, validation planning efforts begin. This planning activity often takes place prior to the DoD program decision by the Defense System Acquisition Review Council (DSARC-I). In some cases, RFPs are actually released, proposals submitted, and responses evaluated -- everything short of actually making the award. It is at this time that the budgetary estimate is updated and an Independent Validation Cost estimate is made. The Independent Cost estimate is used to evaluate the reasonableness of the contractor's proposed costs. Planning documents that the software analyst should help prepare include:

- Work Breakdown Structure (WBS) - The WBS should be prepared by analysts who have both a technical grasp and an understanding of the cost proposal evaluation process. Considerable information as to the validation of the contractor's understanding will be revealed if the government asks the contractor to submit his cost proposal to Level 3 of the software WBS.
- Performance Specification - The performance specification identifies the levels of performance which will be required to meet mission needs. Performance levels that can be considered for trade-offs should be noted.
- Statement of Work (SOW) - The SOW identifies all of the design, engineering, administrative, and support tasks that are to be performed over the contract.
- Government Furnished Information (GFI) - GFI includes all information released to the contractor. This includes any algorithms or reusable code that the government may wish to release.
- Government Furnished Material (GFM) - GFM includes all equipment and material that will be furnished to the contractor by the government. This would include computer hardware.
- Contract Data Requirements List (CDRLs) - CDRLs list all documents which are to be deliverable under the contract.
- Source Selection Plan - The source selection plan includes the procedures and criteria for selecting the winning contractor.

Once a program decision has been made, the contract validation award is made. For high risk programs, there may be more than one validation effort. During the validation phase, the software cost analyst should be

preoccupied with suggesting possible hardware/firmware/software cost trade-offs and in contributing to on-going feasibility and risk assessments. The analyst must become familiar with the contractor's proposed approaches so that independent FSD cost estimates can be prepared. If competing approaches vary significantly, it will be necessary to develop independent cost estimates for each approach. If the program is under Design-to-Cost (DTC), the analyst will develop DTC targets for the program and a tentative allocation of that target down to the subprogram level. Additionally, the analyst is active in preparing selected portions of the Computer Resources Integrated Support Plan (CRISP). Once validation efforts are completed and the FSD proposals submitted, the analyst becomes an active participant in the cost proposal evaluation. Life-cycle cost estimates are also updated for presentation to DSARC.

Once the development decision has been ratified (DSARC II), the analysts participate in Preliminary Design Review (PDR) activites and in overseeing implementation of the contractor's Cost and Schedule Control System. Prior to the System Program Office (SPO) approval of the contractor's proposal, the analyst should prepare independent cost threshold estimates and compare these with those being considered by the contractor. Once PDR is completed, the analyst will monitor the contractor's cost and deliveries. In addition, the FSD cost estimate will be updated based on the results of PDR. Monthly independent estimates of cost at completion (EAC) will be made and compared with those submitted by the contractor. Significant variances will be called to the SPO's attention. Independent cost estimates will be prepared for each Class I Engineering Change Proposal (ECP) and any Class II ECP specified by the SPO. If the program is a Selected Acquisition Review (SAR) designated program, the analyst will prepare appropriate cost status information for inclusion in that report. As the Installation/Production decision (DSARC III) draws near, the analyst will update his life-cycle cost estimates.

3.2 Developing the independent program cost estimate.

This section presents a framework for estimating software program development cost and schedule. Operations and support costs are not discussed in this guide, but must be considered in preparation of the system's Life-Cycle Cost (LCC) estimate.

3.2.1 Development schedule. Table 7 summarizes program development times which are considered reasonable for each development phase. Lower values of the estimators reflect programs that are well planned, well supported by higher authority, and not overly complex. The higher values are more typical of those currently expected because of the trends towards greater system complexity. In the event that the software development is not affected by the DSARC process, the program decision time increments are not relevant.

3.2.2 Development cost. The algorithm for software program development cost (C) is given as:

$$C = C_{CF} + C_{VAL} + C_{FSD} \tag{9}$$

where

C_{CF} = Conceptual Phase cost in dollars

C_{VAL} = Validation Phase cost in dollars

C_{FSD} = Full-Scale Development cost in dollars

TABLE 7: SOFTWARE DEVELOPMENT SCHEDULE ESTIMATORS

Development Phase	Estimator, Months
1. Conceptual Phase	
(a) Concept definition	6 to 24
(b) Program decision	3 to 9
2. Validation Phase	
(a) Solicitation/award	6 to 9
(b) Contract definition[1]	$6 \leq .35\ D \leq 18$[2]
(c) Ratification decision	3 to 6
3. Full Scale Development Phase	$D = \frac{1000\ I}{99.25 + 233(I)^{.667}}$

1. Contract definition is that phase of development during which preliminary design is verified or accomplished, and firm contract and management planning are performed.
2. In general, contract definition will last approximately 35 percent of the total software development time. However, it is estimated that a minimum of six months and a period of no more than eighteen months would be required.

3.2.2.1 Government cost. Table 8 summarizes level of effort estimators that may be used for determining the government's approximate costs during development. These costs cover the program office and those personnel that may be in direct support of the program office. Aron[11] noted that the Conceptual Phase usually requires up to seven people. No people may be assigned in cases where the project is a logical follow-on to an existing activity. Decisions in this case are made by AFSC Project Division Managers as a normal part of their job. The upper limit of seven reflects the fact that, even for large systems, the system concept should be within the grasp of one person. With more than seven, it becomes almost impossible for the team to arrive at a single concept they all understand. Typically the team consists of four to five people. Only a short time is required to prove the concept, but the phase may well last a year or two, during which time missions will be

11. Ibid.

TABLE 8. ESTIMATORS FOR APPROXIMATING GOVERNMENT SOFTWARE DEVELOPMENT COST

Development phase	No. personnel	Man-months	Cost, dollars
1. Conceptual phase			
(a) Concept definition	0 to 7	0 to 168	(0 to 168)(MM)(cost/MM)
(b) Program decision	3 to 7	9 to 63	(9 to 63)(MM)(cost/MM)
2. Validation phase			
(a) Solicitation/award (b) Contract definition (c) Ratification decision	$\frac{(.1 \text{ to } .2)(MM)}{D}$	(.1 to .2)(MM)	(.1 to .2)(MM)(cost/MM)
3. Full Scale Development (FSD) Phase	$\frac{(.2 \text{ to } .4)(MM)}{D}$	(.2 to .4)(MM)	(.2 to .4)(MM)(cost/MM)

See Section 2 for man-month (MM) and duration (D) algorithms.

defined, organizational attitudes will be surveyed to determine needs, and management support will be lined up.

During the validation phase the project office effort is estimated to be 10 to 20 percent of the effort that will be expended by the contractor during FSD. During FSD project office expenditures can be expected to double.

3.2.2.2 Contractor costs. Table 9 summarizes the Contractor Development Activities level of effort estimators for both the Validation and Full-Scale Development Phases.

The effort required of a single contractor during validation is estimated to range from approximately 10 to 20 percent of the contractor effort during FSD. If there is more than one competitive contract definition contractor during validation, the cost should be increased accordingly. If Independent Validation and Verification (IV&V) is to be considered, add another 20 percent to the contractor's FSD costs.

3.3 Cost and schedule control considerations

Major hardware developments are monitored in accordance with DoD Instructions 7000.2 and 7000.10, as well as AFSCP 173.5. Figure 11 illustrates the type of information reported monthly by the contractors for each WBS element. The system forces the contractor to report when agreed-to-cost and schedule thresholds are broken. In addition, the contractor must identify corrective actions being taken. Figure 12 illustrates an effective graphic means for plotting cost and schedule variances (+ variances are good, - variances are bad; by convention).

For software development projects, it is possible to impose a similar reporting scheme on the contractor. This would require that the WBS be both end-item (software subprograms, documentation, etc.) and functionally (coding, package test, etc.) oriented. It is recognized that initially all subprograms may not be identified; however, as the subprograms are identified, they should be included in the WBS.

TABLE 9. ESTIMATORS FOR APPROXIMATING A CONTRACTOR'S SOFTWARE DEVELOPMENT COST

Development phase	Average no. personnel	Man-months	Cost, dollars
1. Conceptual phase (a) Concept definition (b) Program decision	Typically, contractor efforts during the Conceptual Phase are at no cost to the government.		
2. Validation phase (a) Solicitation/award (b) Contract definition (c) Ratification decision	$\frac{(.1 \text{ to } .2)\ MM\ (N)}{D}$	(.1 to .2)(MM)(N) where N = number of competitive contract definition contractors	(.1 to .2)(MM)(N)(cost/MM)
3. Full Scale Development (FSD) Phase	$\frac{MM}{D}$	MM (1 + IVV) where IVV = 0 or .2	(MM)(1 + IVV) (cost/(MM)

See Section 2 for man-month (MM) and duration (D) algorithms.

CLASSIFICATION

Page ___ of ___

CONTRACTOR: LOCATION: RDT&E ☐ PRODUCTION ☐	COST PERFORMANCE REPORT – WORK BREAKDOWN STRUCTURE			SIGNATURE, TITLE & DATE	FORM APPROVED O.M.B. NUMBER 22R0280
	CONTRACT TYPE / NO.:	PROGRAM NAME NUMBER	REPORT PERIOD:		

QUANTITY	NEGOTIATED COST	EST COST AUTH. UNPRICED WORK	TGT PROFIT/FEE %	TGT PRICE	EST PRICE	SHARE RATIO	CONTRACT CEILING	EST CEILING

ITEM	CURRENT PERIOD					CUMULATIVE TO DATE					AT COMPLETION		
	BUDGETED COST		ACTUAL COST WORK PERFORMED	VARIANCE		BUDGETED COST		ACTUAL COST WORK PERFORMED	VARIANCE		BUDGETED	LATEST REVISED ESTIMATE	VARIANCE
	Work Scheduled	Work Performed		Schedule	Cost	Work Scheduled	Work Performed		Schedule	Cost			
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)
WORK BREAKDOWN STRUCTURE													
GEN AND ADMIN													
UNDISTRIBUTED BUDGET													
SUBTOTAL													
MANAGEMENT RESERVE													
TOTAL													
RECONCILIATION TO CONTRACT BUDGET BASELINE													
VARIANCE ADJUSTMENT													
TOTAL CONTRACT VARIANCE													
				(ALL ENTRIES IN THOUSANDS OF DOLLARS)									

Format 1

CLASSIFICATION

Figure 11. Cost and Schedule Control Report Format

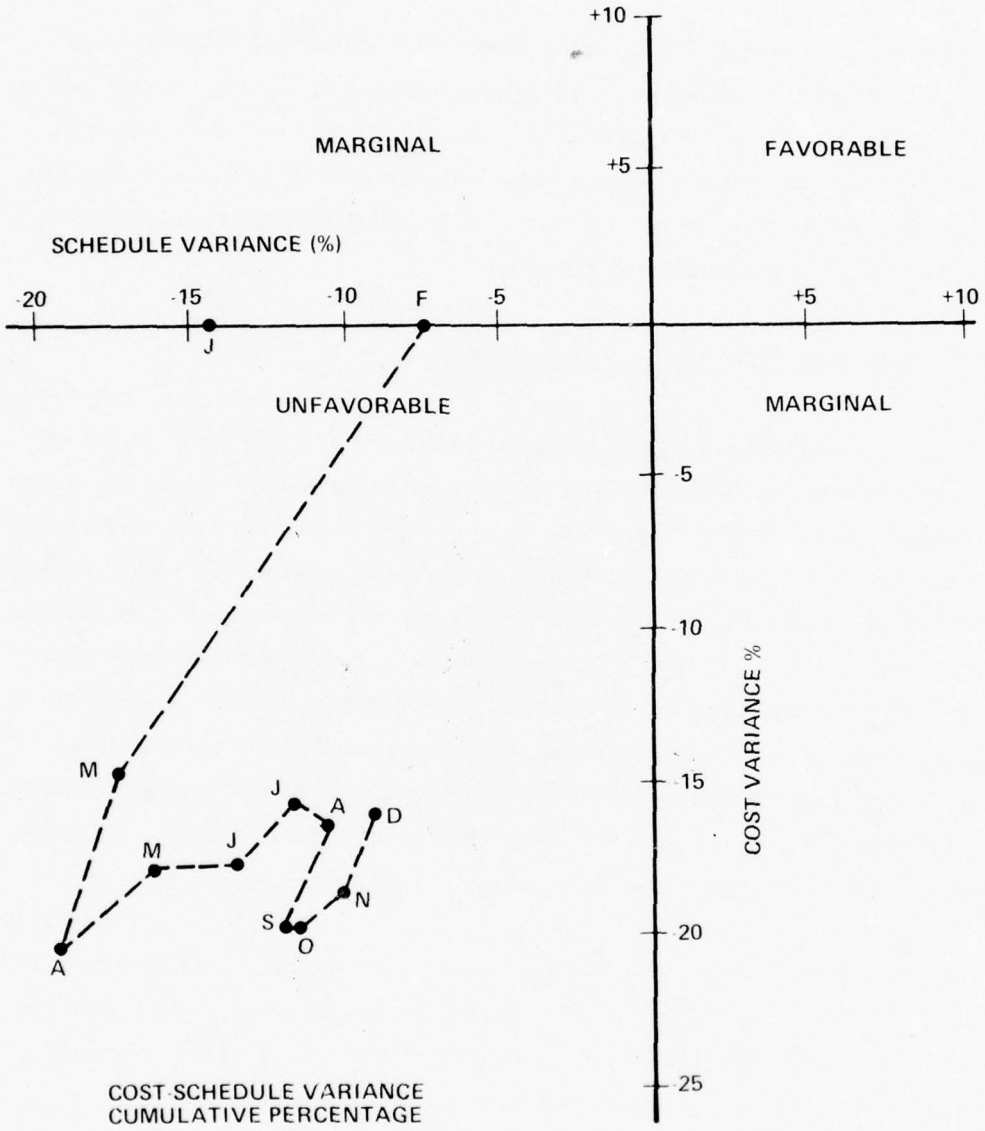


Figure 12. Illustrative Cost & Schedule Variances[12]

12. U.S. Army Management Engineering Training Agency, "Status, Trends, and Projections", January 1975.

4. CONSIDERATIONS IN MANAGING A MAJOR SOFTWARE DEVELOPMENT PROGRAM

Previous sections of this guide have been addressed to the Program Manager's Software Specialist. This section speaks primarily to the Program Manager. It covers pitfalls that should be avoided, use of Independent Cost Estimates, techniques for successfully monitoring the development, and a discussion of factors which will materially affect the magnitude of the development effort.

4.1 Software development problem areas

Figure 13 summarizes those problem areas that are likely to be encountered throughout the software development program.[13] Problem areas are grouped horizontally under the appropriate DoD Directive 5000.1 Development Phase. The flow lines are intended to represent cause/effect relationships among the problems identified. Problem areas are further grouped vertically into one of three domains, e.g., Requirements Domain, System Architecture/Engineering Domain and Management Domain. Appendices A, B, and C provide discussions of specific factors within these domains which materially affect the magnitude of the development effort, and guidelines for responding to the effects of its factors. Table 10 summarizes the sensitivity of development costs to these factors.

The large number of problems that arise during the later stages of Program Validation and during Full-Scale Development is seen to be the result of the combined impact of more stringent and growing requirements on system performance, and the relatively undeveloped state of software design and control methodology. The manifestations of these problems are numerous. In the recommendations that follow most of

---

13. Kossiakoff, A., et al. "DoD Weapon Systems Software Management Study", Johns Hopkins University Applied Physics Laboratory, June 1975.

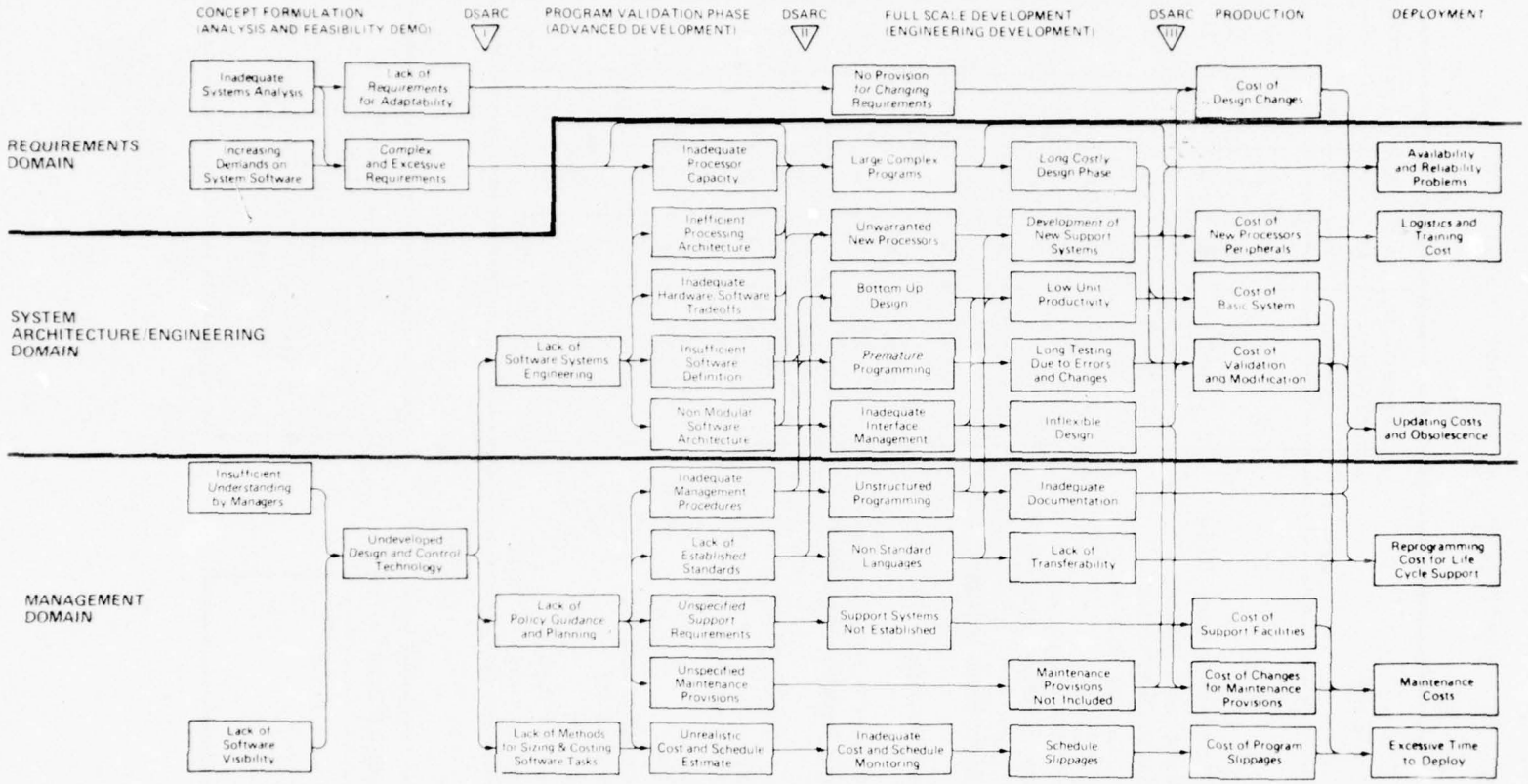


Figure 13. Software Development Problem Areas[14]

14. Ibid.

TABLE 10. SENSITIVITY OF SOFTWARE COSTS TO FACTORS

Factor	Appendix reference page	Software application				
		C&C	Sc.	Bs.	Ut.	All
Communication	C-25	H	H	H	H	H
Constrained, CPU Time	B-2	M	H	N	H	M
Constrained, Program Memory Size	B-4	M	M	M	L	M
Constrained, Time and Memory	B-7	H	H	H	H	H
Cost of Secondary Resources	C-15	N	N	N	N	N
Cost/Schedule Control Systems Criteria (C/SCSC)	C-8	N	N	N	N	N
Data Management Techniques	C-19	M	M	M	M	M
Data Collection, Amount and Method of	C-13	N	N	N	N	N
Developer's First Time on Specified Computer	C-35	H	H	H	H	H
Developer Using Another Activity's Computer	C-30	M	M	M	M	M
Development of Hardware, Concurrent	C-33	H	H	M	M	H
Development and Target Computer Different	C-23	H	L	M	N	M
Development Personnel Mix	C-10	M	M	M	M	M
Development Site	C-29	M	M	M	M	M
Development Site, Number of	C-31	M	M	M	M	M
Design Complexity	B-10	H	H	L	L	M
Design Stability	B-9	H	H	H	H	H
Instruction, Definition of	C-16	H	H	H	H	H
Innovation, Degree of	C-5	H	H	H	H	H
Programmer Testing	C-11	H	H	H	H	H
Programming Facilities	C-22	H	H	H	H	H
Programming Techniques, Modern	C-21	H	H	H	H	H

APPLICATION LEGEND
C&C = Command and Control
Sc. = Scientific
Bs. = Business
Ut. = Utility

SENSITIVITY LEGEND
H = High significant impact
M = Medium significant impact
L = Low significant impact
N = Negligible impact

TABLE 10. SENSITIVITY OF SOFTWARE COSTS TO FACTORS (Continued)

Factor	Appendix reference page	Software application				
		C&C	Sc.	Bs.	Ut.	All
Requirements, Language	C-27	H	H	H	H	H
Requirements, Maintainability	A-16	H	H	H	H	H
Requirements Changes, Operational	A-5	H	H	H	H	H
Requirements Definition, Operational	A-2	M	H	N	N	L
Requirements/Design Interface, Operational	A-7	H	H	H	H	H
Requirements, Quality	A-17	H	H	H	H	H
Requirements, Reliability	A-15	M	M	M	M	M
Requirements, Special Display	C-36	L	L	M	N	L
Requirements, Testing Including V&V	C-6	M	M	M	M	M
Requirements, Transportability	A-22	M	M	M	M	M
Requirements, User Considered	A-6	M	M	M	M	M
Sites, Multiple Software Utilization	C-31	M	M	M	M	M
Sizing Error	C-18	H	H	H	H	H
Software Development Schedule	C-37	H	H	H	H	H
Support Software Availability	C-2	H	H	H	H	H
Specified Response Time	A-8	M	H	H	N	M
Target CPU Designation	B-8	H	H	H	H	H
Work Breakdown Structure	C-3	H	H	H	H	H

APPLICATION LEGEND
C&C = Command and Control
Sc. = Scientific
Bs. = Business
Ut. = Utility

SENSITIVITY LEGEND
H = High significant impact
M = Medium significant impact
L = Low significant impact
N = Negligible impact

the suggested approaches lie either in the domain of Software System Architecture/Engineering or Management, which are different, but related, approaches to the development of improved software design and control methodology.

An important consideration noted in the flow line at the top of Figure 13 is the inevitable growth and change of requirements throughout a system's lifetime. For weapon systems, these changes are inherent in the changing nature of the threats against which most systems must operate, as well as in the fact that software can be modified without physical changes to the system. In practice, unless provisions for adaptation to change are designed into a system, the consequences are often serious.

4.2 Areas requiring the software cost analyst's input

Independent cost estimates, prepared by the team's software cost analyst, serve an increasingly important role throughout the software development program. Over the life of the development program, it is reasonable to assume that at least four different estimates will be prepared. With each succeeding estimate, additional, more definitive information becomes known, thereby decreasing the uncertainty associated with the estimate. The four independent estimates are, in the order of their occurrence:

Estimate No. 1 - Initial Program Budgetary Estimate

Estimate No. 2 - Independent Cost Estimate of Program Validation

Estimate No. 3 - Independent Cost Estimate of Full-Scale Development (FSD)

Estimate No. 4 - Update of FSD Cost Estimate

Figure 10 (p. 31) provides a checklist of technical and financial documents and/or events requiring inputs from the software cost specialist. These documents/events should be well-known to the Program Manager.

It is suggested that this list be reviewed and that a continuing dialogue be developed with the software program cost analyst. In most instances this individual, or group of individuals, will be qualified both in technical and costing aspects of the software program.

4.3 Monitoring the development

The discussion which follows is oriented towards the Air Force 800 series regulation and manuals covering research, development, engineering, testing and production of systems of all types. However, to a certain degree, the discussion is also applicable to the 300 series. The cost estimates developed in prior sections are relevant.

Each major development phase is discussed in turn. Questions requiring answers are asked. Do's and Don'ts are listed as appropriate.

4.3.1 Conceptual phase. The objectives of this phase are to develop a system concept, examine trade-offs and conduct feasibility assessments. The principal output of this phase is the initial system specification which establishes the functional baseline. Typically, according to Aron,[15] the conceptual team consists of four to five people, but could range from one to seven. In some cases where the project is a logical follow-on to an existing activity, the activity staff will perform the functions of the conceptual team. Decisions in this case are made by AFSC Project Division Managers as a normal part of their job. Aron has commented that there is an upper limit of seven above which it becomes almost impossible for the team to arrive at a single concept they all understand. Although a short time may only be required to define a

---

15. Aron, op. cit.

concept, the entire conceptual phase may last a year or two during which mission requirements are delineated and appropriate approval of the concept obtained.[16]

QUESTIONS WHICH SHOULD BE ASKED

a. Has user involvement been a part of conceptual definition?

b. Have software development risks been identified?

c. Will the software development project be assigned to a single contractor or will more than one contractor be involved? If more than one contractor, how will responsibilities be assigned?

d. What are the technical requirements of the contracts?

e. Have testing requirements been established? Will Validation and Verification be accomplished by an independent contractor or by the project office?

f. Has there been an adequate evaluation of software versus hardware trade-offs?

SOME DO'S

a. Insist on a clear definition of operational requirements (preferably documented).

b. Achieve user participation in concept definition.

c. Require an analysis of software development risks.

d. Identify total software life-cycle requirements.

e. Require the development of plans for the orderly acquisition of the software such as a Computer Program Development Plan (CPDP) [DI-E(U)695/ESD] and a Computer Resources Integrated Support Plan (CRISP), which will ensure that the life-cycle requirements are satisfied.

---

16. Ibid.

4.3.2 Validation phase. The objectives of this phase are definition and validation of the system requirements. The principal output of this phase is the development specification. To achieve this, competitive Contract Definition efforts are desirable wherever feasible.

Validation is accomplished by conducting a thorough requirements analysis which examines trade-offs between general and special purpose computers, determining which computer and peripherals should be utilized for specific applications. This analysis is made during the architectural design of the general and special purpose computers and their associated software. During this phase, requirements for each software package as well as delineation of all external interface requirements should be developed.

The next step, Software Design Analysis, is the detailed breakout of each computer software package into functional units or modules. Response time estimates and memory allocations should be determined through preliminary design, modeling and simulations, and as a result, a Design Criteria statement for each software package should be created. As each step is accomplished, a more definitized functional detailing of each unit or module will be achieved. In this step by step approach specifications for each unit or module will be completely and accurately defined. As problems are identified, iterations through prior steps are made to resolve them.

SOME DO'S

a. Identify roles and responsibilities of all organizations as early as possible.

b. Consider flexibility in schedule and cost for possible change in operational requirements.

c. Utilize software prototyping and/or parallel development where significant risks or requirements/uncertainties exist.

d. Standardize and disseminate algorithms required by operational requirements by including them in the RFP wherever appropriate.

e. Use separate validation resources.

f. Require the contractor to include a Computer Program Development Plan (AFR 800-14, Volume 2) in his proposal.

g. Be wary of target prices which are significantly less than the Program Manager's independent cost estimate.

h. Continue negotiations until a mutually satisfactory understanding is reached as to the development activity's approach and understanding.

i. Probe proposed subcontractor relationships to find out the extent of agreements between the prime contractor and the subcontractors with respect to responsibilities, technical performance and prices.

SOME DON'TS

a. Permit language proliferation. Keep it constrained or it will increase software development costs.

b. Forget to review the response time characteristics of the system.

c. Hesitate to specify the Central Processing Unit (CPU) once the validation of hardware/software/firmware trades have been completed.

4.3.3 Full-Scale development phase. The full-scale development phase encompasses analysis and design, coding and checkout, and system test and integration. Analysis commences with the release of the development specification and terminates with the successful accomplishment of System Development, Test, and Evaluation (DT&E) or Software Functional Qualification Test (FQT).

During this phase, various design approaches are considered, analyses and trade-offs performed, and design approaches selected. The purpose of the design phase is to develop a design approach including mathematical models, and functional or detail flow charts, if required/desired.[17,18] The design approach should also define the relationship between the computer program components. This information is contained in the preliminary computer product specification and is normally presented and reviewed during the Critical Design Review (CDR). Coding and checkout commences with the successful accomplishment of CDR. Test and integration compares the program results against the requirements specified in the computer program development specification. This test and integration process includes the individual computer program function or module tests, and extends through total computer program formal qualification tests.

QUESTIONS WHICH SHOULD BE ASKED

a. Does the FSD program include provision for adequate modern support tools and facilities, including such items as assemblers, compilers, editors, debug aids, data base and library management systems, and associated operating systems?

b. Have time, resources, and testing aids been properly allocated to permit iterations of unit or module design in the overall software program planning schedule?

c. What precautions have been developed to alert management of emerging problem trends? What checkpoints have been established in the overall software program planning schedule?

d. Has a formal software Quality Assurance (QA) program been established by the contractor?

e. Are there provisions for validating and verifying the software development activity and cost by someone other than the development contractor?

---

17. Thomas M. Kraly, et al., "Structured Programming Series, Volume VIII: Program Design Study." IBM Corporation, RADC-TR-74-300, May 1975.

18. L. H. Ortega, "Structured Programming Series, Volume VII: Documentation Standards." IBM Corporation, RADC-TR-74-300, September 1974.

f. Have provisions been made to assure delivery of the necessary support software, system resources, and related documentation to satisfy operations and maintenance support functions?

SOME DO'S

a. Establish specific development milestones for software programs.

b. Establish specific decision points during the software development phase.

c. Require reporting of specific software management information and thresholds.

d. Impose a design freeze to the maximum extent possible after the design reaches sufficient maturity.

e. Monitor early testing to validate the contractor's (often overly optimistic) progress estimates.

f. Provide fast response to the development activities' action requests.

g. Require periodic and open Quality Assurance (QA) reviews with the contractor.

h. Require the utilization of developed support software or the creation of support software prior to the development of the primary software packages. Discourage the concurrent development of support software.

i. Plan adequate time and resources for design and design iterations.

j. Progressively test each unit or module software package as it is completed, as well as the interfaces between completed modules.

k. Begin configuration control of the allocation baseline immediately before PDR.

SOME DON'TS

a. Permit the development of computer software and hardware concurrently unless the overall software program plan includes time and resources for both software and hardware design iterations.

b. Divide responsibility for concurrent, related software development among several different development activities.

c. Hesitate to impose a formal cost and schedule requirement on the developer, wherever appropriate.

d. Develop the software at more than one location unless the cost and schedule impacts are acceptable.

APPENDIX A

DISCUSSION OF REQUIREMENTS DOMAIN FACTORS

FACTOR IMPACT CONVERSION

In this appendix, many of the factor impacts are presented as they affect productivity. In order to convert this impact to a cost multiplier for use in the algorithms proposed in this guide, you need only do the following:

- If the factor impact causes a decrease in productivity, subtract the percent decrease from 100 percent and divide the remainder into 100 percent. For example, if the factor effect of a command and control application amounts to a 20 percent decrease in productivity which accordingly increases the costs of the program, the cost multiplier can be obtained by subtracting 20 percent from 100 percent, and then dividing the remainder (80 percent) into 100 percent. The calculations will result in a cost multiplier of 1.25.
- If the factor impact causes an increase in productivity, you would simply add the percent increase to 100 percent and then divide into 100 percent to obtain the cost multiplier for the algorithms.

REQUIREMENTS DOMAIN FACTOR NO. 1
OPERATIONAL REQUIREMENTS DEFINITION

QUESTION: How clearly have operational requirements of the Performance Specifications been defined?

GENERAL IMPACT: For systems which have operational requirements defined only vaguely or in outline form, one can expect to pay significantly more, depending on type of program being developed (see Effect on Productivity on page A-4), than for systems which have operational requirements well defined.

GUIDELINES:

- Make sure that sufficient detail gets reflected in the requirements analysis portion of software development prior to design.
  - Identify and delineate all software (S/W) functions.
  - Define the operational characteristics of S/W and operational constraints of S/W.
  - Questions to be considered:
    - Have S/W algorithms been developed?
    - Have hardware (H/W) and S/W trade-offs been made?
    - Have firmware (F/W) and S/W trade-offs been made?
    - Have functional alternatives to Computer Program Configuration Items (CPCIs) been made?
    - Have all requirements for test and evaluation been established?
    - Has the User activity assisted and approved the operational requirements definition?
- Approaches that can be taken to ensure the requirements analysis is adequate are as follows:

REQUIREMENTS DOMAIN FACTOR NO. 1
OPERATIONAL REQUIREMENTS DEFINITION (Continued)

GUIDELINES: (Continued)

- Determine if there is a formal requirements analysis language available for the application area involved. There are, for example, machine resident languages available for application areas like Ballistic Missile Defense. The output of such a language is input to design, again usually a formal language. If such a language is available, then one can use it to assure that a sufficient level of detail is attained in requirements analysis. However, the use of the language will not automatically guarantee the level of detail required.
- Determine if an existing requirements analysis language can be adapted to the application area involved. If yes, then one can expect to reap the benefits such a language provides as stated above. However, the adaptation of the language will require additional development dollars, and most likely, an initial schedule delay. For large developments, the increased cost may be recouped and further slippage abated through the use of the language. Assess if the language is of use in other developments. If yes, then one may be willing to accept increased cost on one development if cost will be lessened for succeeding developments. Determine if the cost of adaptation is absorbed by a support activity, such as Rome Air Development Center (RADC) for Electronic Systems Division (ESD) SPO's.
- Determine if a requirements analysis language can be written for the application area involved. If yes, then the same trade-offs have to be examined as described above for adaptation. However, the cost of developing a language from scratch will probably be greater than adapting an existing language.

REQUIREMENTS DOMAIN FACTOR NO. 1
OPERATIONAL REQUIREMENTS DEFINITION (Continued)

GUIDELINES: (Continued)

- Determine what non-formal procedures can be implemented in requirements analysis to assure adequate detail. There are no hard and fast rules available, but the project manager should keep requirements analysis high on his awareness scale. Don't let design start until both the purchaser and developer feel comfortable with the requirements.

- Consideration should be given to how requirements analysis is paid for, especially algorithm development. It can be by the purchaser or developer, and it may or may not be delineated as a software cost. Such considerations should be taken into account when structuring the WBS and making cost estimates.

VAGUE OPERATIONAL REQUIREMENTS	
EFFECT ON PRODUCTIVITY:	
COMMAND & CONTROL:	35% DECREASE
SCIENTIFIC:	50% DECREASE
UTILITY:	NO EFFECT*
BUSINESS:	NO EFFECT*
ALL OF THE ABOVE:	10% DECREASE

* The implication here is that the operational requirements of utility and business programs are usually defined adequately prior to design. Thus, no effect was noted.

REQUIREMENTS DOMAIN FACTOR NO. 2
OPERATIONAL REQUIREMENTS CHANGES

QUESTION: How can the effect of changes in operational requirements of the Performance Specifications be assessed?

GENERAL IMPACT: Costs were observed to be considerably higher for systems which have undergone requirements changes as compared to those with frozen requirements. The impact, however, will be highly dependent on the individual case.

GUIDELINES:

- Ensuring that the original operational requirements are adequate can prevent some changes.
- Perform tradeoffs between the costs and benefits of change vs. no-change, and among alternative changes.
- Do not expect to go through a large development without changing requirements; therefore, provision for change should be made.
- A requirements change will often result in both the discarding of some existing code and the addition of new code. Sometimes, however, only the latter is involved. In either case, expect both the cost to increase and the schedule to slip due to the requirements change. Each change should be addressed independently in terms of the amount of code to be written to accommodate the change. The amount of code to be written for the change can then be used to assess the cost and schedule impact.
- Try to discourage changes, especially major ones, particularly after the design phase has been completed.

CHANGES IN OPERATIONAL REQUIREMENTS	
EFFECT ON PRODUCTIVITY:	
AVERAGE:	5% DECREASE
MAXIMUM:	95% DECREASE

REQUIREMENTS DOMAIN FACTOR NO. 3
USER REQUIREMENTS CONSIDERED

QUESTION: To what degree are users involved in the development?

GENERAL IMPACT: The impact of not obtaining user participation in early concept definition can cause a cost increase of as high as 100 percent.

GUIDELINES:

- The project officer or SPO should be completely aware that the development will be deemed unacceptable if it does not meet user requirements. Therefore, the end user should be involved as much as possible in the development.
- The Air Force has guidelines and procedures in AFSCP 800-3 to ensure that end user requirements are input properly to the developer, via the SPO.
- Insure that the user is involved in the development of the operational requirements and that the user is aware of all changes to the requirements and design.

LITTLE OR NO USER PARTICIPATION IN DEVELOPMENT
EFFECT ON COST: SIGNIFICANT INCREASE

REQUIREMENTS DOMAIN FACTOR NO. 4
OPERATIONAL REQUIRE- MENTS/DESIGN INTERFACE

QUESTION: How well do operational requirements interface to design?

GENERAL IMPACT: The impact of this factor has not been measured quantitatively, but is considered to be large.

GUIDELINES:

- A very detailed requirements analysis should be accomplished to ensure that the operational requirements accurately interface with the design. This can usually be done by using one of the two following alternatives:

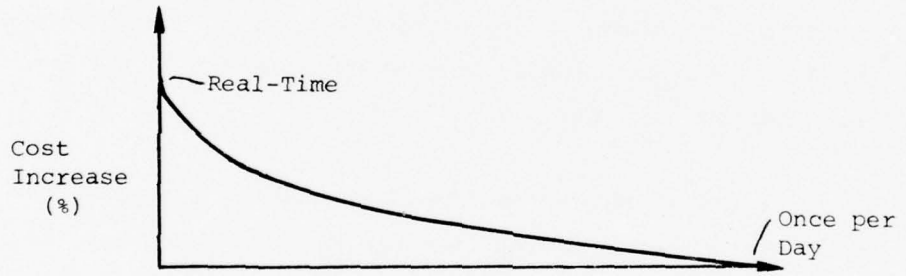
  - Impose a formal machine resident requirements analysis language and companion design language on the development, if available. The advantages and disadvantages of this alternative are covered under Factor No. 1 - Definition of Operational Requirements, of this Appendix.
  - Have representatives of the intended software developer and the user participate in the requirements analysis.

INACCURATE INTERFACING
EFFECT ON PRODUCTIVITY: SIGNIFICANT DECREASE

REQUIREMENTS DOMAIN FACTOR NO. 5
SPECIFIED RESPONSE TIME

QUESTION: What are the response time characteristics of the system?

GENERAL IMPACT: Software that has to respond in real time will cost more to develop depending on the program application (see Effect on Productivity on Page A-11). The curve below represents impact as a function of response time.



GUIDELINES:

- Factor the software into response time domains, at least to the point of getting the real-time dependent portion clearly isolated. Use the following guidelines once the total software development has been partitioned according to response time:
  - If the real-time dependent portion represents less than 10 percent of the total development, then no special provisions are necessary. Just expect lower programmer productivity for that portion of the development.

REQUIREMENTS DOMAIN FACTOR NO. 5
SPECIFIED RESPONSE TIME (Continued)

GUIDELINES: (Continued)

- If the real-time dependent portion represents more than 10 percent of the total development, the the following alternatives should be considered:
  - Consider a hardware trade-off to minimize costs while satisfying performance requirements. If only one system is being developed, the additional cost of hardware will probably be less than the high cost of software (caused mainly by rewrites) to satisfy the response time requirement. If more than one system is being installed in the field, attempt to determine the break-even point in number of systems. If the projected number to be installed is less than the break-even point, consider the hardware alternative. If the projected number to be installed is greater than the break-even point, consider leaving the intended task in software.
  - Consider a firmware trade-off. Software still has to be developed for this alternative since all firmware originates as software. Software targeted for firmware is generally referred to as microcode. Writing microcode to meet a response time requirement will be less productive per line of source code than ordinary software, but will not generally require the number of rewrites that meeting a tight response time requirement in ordinary software would require, since the microcode usually resides in a faster memory. Depositing critical functions in firmware will relax Central Processing Unit (CPU) time loading.

REQUIREMENTS DOMAIN FACTOR NO. 5
SPECIFIED RESPONSE TIME (Continued)

GUIDELINES: (Continued)

The firmware option will require an additional hardware cost for each system, since a high speed memory will be required for the microcode to reside in. However, this increased hardware cost per system will not be as large as the total hardware option examined above. For a single installation development, depositing critical response time functions in firmware will probably pay off. For multi-installation developments, use the guidelines presented above for the hardware options for firmware tradeoffs.

- Examine the use of a faster, more powerful CPU. This will increase hardware costs, and also impinge on weight and volume constraints, if applicable. The guidelines presented in the hardware trade-off above apply.
- Consider a multiple CPU system. This is a possibility if several real-time dependent functions are vying for CPU service simultaneously in an interrupt driven system. The CPU time loading can sometimes be relieved by spreading the functions across multiple CPUs. If, however, one function is the driving factor, then this is not a viable alternative. This alternative will increase hardware costs, and also impinge on weight and volume constraints, if applicable. The guidelines presented in the hardware trade-off above apply.
- Obtain relaxation and/or removal of real-time requirements (obtain and document user concurrence).

REQUIREMENTS DOMAIN FACTOR NO. 5
SPECIFIED RESPONSE TIME (Continued)

GUIDELINES: (Continued)

- If required, accept a reduction in programmer productivity with concomitant increase in software costs.

- This is a factor which is very important for on-board flight programs in avionics software developments. It is much less important in command and control applications.

REAL-TIME REQUIREMENT	
EFFECT ON PRODUCTIVITY:	
COMMAND & CONTROL:	25% DECREASE
SCIENTIFIC:	40% DECREASE
UTILITY:	70% DECREASE
BUSINESS:	NO EFFECT
ALL OF THE ABOVE:	25% DECREASE

REQUIREMENTS DOMAIN FACTOR NO. 6
AVIONICS APPLICATION

QUESTION: Is the software being developed for an avionics application?

GENERAL IMPACT: Less programmer productivity than for other types of software in general, but magnitude depends on mix of on-board flight programs, simulation, and Automatic Test Equipment (ATE).

GUIDELINES:

- Partition software into three categories:
  - On-board flight programs,
  - Simulation, and
  - ATE.

  On-board flight programs will be least productive because of time and memory constraints, and extensive testing. Simulation will be second least productive, and ATE most productive. Use separate estimators for each category.
- Do not expect a high degree of High Order Language (HOL) implementation for on-board flight programs; however, expect HOL use to increase in the future. Expect more for simulation and ATE.
- A convenient size estimator exists for ATE. Expect about 1500 lines of source code for each Line Replaceable Unit (LRU). This is the smallest unit that can be removed from the aircraft. Large aircraft will generally have more LRUs than small aircraft. A mid-range size is about 150 LRUs.
- A constrained memory could cause serious problems.

AVIONICS APPLICATION
EFFECT ON PRODUCTIVITY: DECREASE

REQUIREMENTS DOMAIN FACTOR NO. 7
COMMAND & CONTROL APPLICATION

QUESTION: Is the software being developed for a command and control application?

GENERAL IMPACT: About 40 percent less programmer productivity on the average than for all other software applications, in general.

GUIDELINES:

- The applications are usually large, about 500,000 object words on the average. This size will decrease programmer productivity.
- Most software will be developed on large main frames targeted for large main frames. Therefore, the potential for excellent support software exists, especially for main frames that have been in the field for a long time.
- Most applications can be implemented with a high degree of HOL usage. The standard Air Force language is JOVIAL. Developers who propose a small degree of HOL usage should show cause why.
- Due to size and complexity of design in command and control applications, extra special attention should be directed at accurately defining the operational requirements, user involvement, and an acute awareness that changes can cause reprocussions throughout the overall program package.

COMMAND AND CONTROL APPLICATION
EFFECT ON PRODUCTIVITY: DECREASE

REQUIREMENTS DOMAIN FACTOR NO. 8
MULTIPLE SOFTWARE UTILIZATION SITES

QUESTION: Will the developed software have more than a single site installation?

GENERAL IMPACT: On the average, developing software for a multi-site installation is 30 percent less productive than developing software for a single site installation.

GUIDELINES:

- If the multi-site software to be developed has no site dependent features, then expect no impact from this factor.
- If the multi-site software to be developed has site dependent features, then expect the cost to increase by the number and size of such features to be implemented.
- If the multi-site software to be developed has inter-machine communication, expect the cost per unit line of code delivered to be higher than if no inter-machine communication is required.

MULTIPLE SITE DEVELOPMENT
EFFECT ON PRODUCTIVITY: 30% DECREASE

REQUIREMENTS DOMAIN FACTOR NO. 9
RELIABILITY REQUIREMENTS

QUESTION: How much reliability is required for the delivered software?

GENERAL IMPACT: Higher reliability means higher development costs, but lower maintenance costs. However, the exact quantitative nature of this trade-off is not known.

GUIDELINES:

- There are currently no standard accepted definitions of software reliability.
- Meeting reliability requirements, by whatever definition used, affects the cost in the testing and integration phase of development. The higher the reliability, the higher the cost for testing and integration.
- Break up the total development into reliability categories. Expect higher cost per unit line of code delivered for high reliability categories than for low reliability categories. Assess the reliability required in terms of the failure rate that can be tolerated operationally for each category.

HIGHER RELIABILITY
EFFECT ON COST: INCREASE

REQUIREMENTS DOMAIN FACTOR NO. 10
MAINTAINABILITY REQUIREMENTS

QUESTION: Are maintainability requirements to be imposed on the development?

GENERAL IMPACT: Imposition of maintainability requirements will increase development costs, but decrease maintenance costs. The impact is not easily quantifiable, but is considered highly significant.

GUIDELINES:

- Maintainability is largely a function of the following factors discussed on other guidesheets. Specifically:
  - Language requirements,
  - Reliability,
  - Testing requirements,
  - Transportability, and,
  - Complexity.
  - The most important of these is language. High Order Language (HOL) is much more maintainable than Machine Oriented Language (MOL). Therefore, try to get as much of the development as possible implemented in HOL.
- Another factor affecting maintainability is documentation. Adequate manuals and run sheets for the programs directly affect maintainability. Consider that approximately 30 pages of documentation per 1000 lines of source code delivered will be required.
- If costs are too high, determine if maintenance requirements can be relaxed.

INCREASED MAINTAINABILITY
EFFECT ON COST: INCREASE DEVELOPMENT COSTS

REQUIREMENTS DOMAIN FACTOR NO. 11
QUALITY REQUIREMENTS

QUESTION: What sort of quality requirements are being imposed on the development?

GENERAL IMPACT: The imposition of quality requirements will increase development costs, but decrease maintenance costs. The magnitude of the impact is not known, but considered to be significant.

GUIDELINES:

- An accepted set of attributes of software quality is:[19]
  - Correctness,
  - Reliability,
  - Efficiency,
  - Integrity (security, etc.),
  - Usability,
  - Maintainability,
  - Testability,
  - Flexibility,
  - Portability,
  - Reusability, and,
  - Interoperability.
- Correctness will be assessed in the Verification and Validation (V&V) portion of testing. V&V will increase testing and integration costs, but decrease maintenance costs. Increasing correctness is akin to increasing reliability.
- Increasing reliability will increase testing and integration costs, but decrease maintenance costs. See Factor No. 9 - Reliability, in this appendix.
- Increasing software operating efficiency is a very costly proposition, since it usually requires rewrites to increase efficiency or going to an MOL. It also has a negative effect on most other quality factors, thus increasing life-cycle costs. Since CPU time and memory constraints usually imply the necessity for efficient coding, the cost impacts of these factors also reflect

19. Richards, P.K., et al., "Factors in Software Quality." General Electric Company Presentation under RADC Contract F030602-76-C-0417, December 1976.

REQUIREMENTS DOMAIN FACTOR NO. 11
QUALITY REQUIREMENTS (Continued)

GUIDELINES: (Continued)

the cost impact of increased efficiency. See Factors Nos. 1, 2, and 4 of Appendix B. Since increased efficiency usually has an adverse impact on life-cycle costs, only attempt to obtain the absolute minimum level required. System growth may cause the efficiency to decrease, violating minimum levels. Anticipated growth should be accounted for in the initial design and requirements, thereby decreasing the likelihood that recoding or other measures will be required during maintenance to attain initially specified efficiency levels.

- Increased integrity will increase the amount of code required to meet the same set of operational requirements. This will increase development costs, but decrease operational costs. Integrity essentially characterizes how sensitive the system is to operator error or system error caused by hardware malfunctions. The project manager has to ask the question, "How much down time due to operator or system error can be tolerated in an operational environment?" If a relatively large amount can be tolerated, then a great amount of integrity is not required in the software design. If only a relatively small amount can be tolerated, then a large amount of integrity should be required in the software design. The amount of integrity required for certain classes of on-board flight software in avionics applications is very high. For simulation and Automatic Test Equipment (ATE) for avionics, it is much less. Command and Control will usually fall between on-board flight programs in avionics and simulation and ATE for avionics in terms of integrity required.

REQUIREMENTS DOMAIN FACTOR NO. 11
QUALITY REQUIREMENTS (Continued)

GUIDELINES: (Continued)

- Usability refers to how well the software satisfies user requirements. See Factor No. 3 - User Requirements, of this appendix.
- Increasing maintainability will increase development costs, but decrease maintenance costs. See Factor No. 10 - Maintainability, of this appendix.
- Increasing testability will increase the amount of code required to meet the same set of operational requirements. This will increase the cost of the analysis and design, and coding and checkout phases of development, but decrease cost in the testing and integration phase. The amount of testability required should be a function of the size of the development. It should increase with size.
- Increasing flexibility, i.e., making the software more adaptable to changing requirements, will increase the amount of code required to meet the same set of operational requirements. This will increase development cost, but decrease maintenance costs. The project manager should analyze flexibility requirements in terms of the expected volatility in operational requirements. If the volatility of operational requirements is expected to be low over the life of the system, then great flexibility is not required. If the volatility of operational requirements is expected to be high over the life of the system, then engineer a large amount of flexibility into the software design.

REQUIREMENTS DOMAIN FACTOR NO. 11
QUALITY REQUIREMENTS (Continued)

GUIDELINES: (Continued)

- Portability refers to the ease with which the developed software can be transferred from one hardware configuration and/or software environment to another. Reusability refers to the ease with which the developed software can be used in other applications. These factors are highly interrelated, and are essentially covered in Factor No. 12 - Transportability, of this appendix. A feature of reusability not covered under transportability is the packaging and scope of the functions developed. This is essentially the modularity put into the design. That is, can a subroutine easily be lifted out and deposited into another development without a lot of awkward interfacing problems? Increasing this kind of modularity will increase development costs, but may decrease development costs on subsequent developments. This feature is also related to the following attribute, interoperability.
- Interoperability refers to the ease with which the developed software can couple/interface with another system. Increasing this attribute will increase development costs, but increase the potential use of the system and also possibly its life. Increasing portability and reusability will increase interoperability; therefore, factor No. 12 on transportability applies. A high use of High Order Language (HOL) will increase interoperability. Other features that will increase interoperability are:

REQUIREMENTS DOMAIN FACTOR NO. 11
QUALITY REQUIREMENTS (Continued)

GUIDELINES: (Continued)

- use of standard widely used communications protocols,
- use of standard character representation such as ASCII, and,
- use of standard 32-bit and 64-bit formats for floating point representation.

- Determine the quality requirements of the software package and incorporate the requirements into the design at the earliest feasible point.

IMPROVED QUALITY
EFFECT ON COST: INCREASE IN DEVELOPMENT COST

REQUIREMENTS DOMAIN FACTOR NO. 12
TRANSPORTABILITY REQUIREMENTS

QUESTION: What transportability requirements are to be imposed on the software to be developed?

GENERAL IMPACT: Increasing transportability, if the language mix remains constant, will increase development costs. Generally, this cost can only be recouped if there is a change of CPUs over the life cycle or the code can be transported to other developments. There are secondary cost benefits in training and documentation by using standard versions of standard languages, which inherently makes the code more transportable.

GUIDELINES:

- Identify other potential uses of the code.
- Assess probability of change in CPU.
- Perform cost tradeoffs to evaluate benefit of transportability.
- Code written in an High Order Language (HOL) is more transportable than code written in a Machine Oriented Language (MOL).
- Code written in a standard version of an HOL is more transportable than code written in a non-standard version.
- Code written in a widely used HOL is more transportable than code written in a less widely used HOL.
- The code required to solve a given problem in a standard version of an HOL will generally be greater than that required in a non-standard version, because the non-standard version is almost always a superset of the standard version, offering the programmer more options in solving the problem.

REQUIREMENTS DOMAIN FACTOR NO. 12
TRANSPORTABILITY REQUIREMENTS (Continued

GUIDELINES: (Continued)

- Since transportability is almost solely a function of language requirements, see Factor No. 17 - Language Requirements, in Appendix C for additional considerations.
- Avionics software is much less transportable than Command and Control software, since so much of it has to be implemented in MCL. Most command and control software can be implemented in a standard version of an appropriate HOL, such as JOVIAL.

INCREASED TRANSPORTABILITY
EFFECT ON COST: INCREASE IN DEVELOPMENT COST

REQUIREMENTS DOMAIN FACTOR NO. 13
BUSINESS APPLICATION

QUESTION: Is the software development a business application?

GENERAL IMPACT: Business applications are more productive per unit line of delivered code than non-business applications.

GUIDELINES:

- Most business applications can be implemented in either COBOL or RPG. It is difficult to justify implementation in an MOL, since efficiency requirements, the major reason for MOL implementation, are seldom severe.
- Since business applications generally have a high degree of I/O relative to computation, sizing or costing algorithms based on the number of I/O items can be quite effective.
- A number of business application programs are written on the basis of transaction oriented processing to update files. In these cases, the number of transactions can serve as an estimator of size and cost.
- Most business applications for the Air Force are implemented under the control of the Air Force Data Systems Design Center (AFDSDC). The primary language used is COBOL. A formalized procedure for development exists, and is documented in Design Center manuals.[20] It covers all life-cycle phases from analysis through operation. A management information system for resource

---

20. Air Force Data Systems Design Center Manual 300-8, Gunter AFS, Alabama.

REQUIREMENTS DOMAIN FACTOR NO. 13
BUSINESS APPLICATION

planning and utilization exists called PARMIS (Planning and Resource Management Information System).

BUSINESS APPLICATION
EFFECT ON PRODUCTIVITY: INCREASE

REQUIREMENTS DOMAIN FACTOR NO. 14
SCIENTIFIC APPLICATION

QUESTION: Is the software development a scientific application?

GENERAL IMPACT: Scientific applications are more expensive per unit line of delivered code than non-scientific applications.

GUIDELINES:

- Most scientific applications, except in a real-time environment, can be implemented in an HOL. The widely used languages oriented around batch development are FORTRAN, ALGOL, PL/I, and JOVIAL. PL/I has the additional advantages of having features which are applicable to business applications. The widely used languages oriented around interactive development are BASIC and APL. It is difficult to justify the use of an MOL for scientific applications in anything other than a real-time environment.
- Probably the largest scientific developments in a non real-time environment are Monte Carlo simulations. If the simulation is event driven, then the number of events can be used to estimate size.
- For real-time scientific applications, see the guide sheets on response time (Factor No. 5), and CPU time and memory constraints (Appendix B, Factors Nos. 1 and 2).

SCIENTIFIC APPLICATION
EFFECT ON PRODUCTIVITY: DECREASE

REQUIREMENTS DOMAIN FACTOR NO. 15
UTILITY APPLICATION

QUESTION: Is the software development a utility application, such as tape to line printer, code conversion, or a sort/merge program?

GENERAL IMPACT: In general, if the software is a utility application, the cost per unit line of delivered code will be less than that of other applications, excepting business applications.

GUIDELINES:

- Developing support software that operates in the utility mode such as converting information from one medium to another (tape to disk, etc.), listing programs, code conversion (ASCII to BAUDOT, etc.), and sort/merges is more productive in terms of cost per unit line of delivered code.
- If this type of software represents less than 10 percent of the total development, there will be no effect on cost or productivity.

UTILITY APPLICATION
EFFECT ON PRODUCTIVITY: INCREASE

APPENDIX B

DISCUSSION OF SYSTEM ARCHITECTURE/ENGINEERING (A/E) FACTORS

FACTOR IMPACT CONVERSION

Refer to note on page A-1

A/E DOMAIN FACTOR NO. 1
CPU TIME CONSTRAINED

QUESTION: Is it projected that the CPU will be in a time constrained mode?

GENERAL IMPACT: If the CPU is projected to operate in a time constrained mode, the cost is expected to increase. A time constrained mode is defined as more than 80 percent utilization of available CPU time for the most demanding task.

ACTION CONSIDERATIONS:

- This factor is highly correlated with the response time factor (see page A-8), since the constraint is most often present in real-time environments. However, not all real-time environments will present the constraint. For example, a mini-computer controlling machine tools will be in a real-time environment, but the time-constraints are not severe. In contrast, navigation, fire-control, and signal processing computers in an avionics subsystem will most definitely be affected by the constraint.
- Factor the software into time constrained and non-time constrained tasks using the 80 percent CPU loading rule. Use the following guidelines after the software has been partitioned in this manner:
  - If the time constrained portion represents less than 10 percent of the total development, then no special provisions are necessary. Just expect lower programmer productivity for that portion of the development.
  - If the time constrained portion represents more than 10 percent of the total development, then use the alternatives presented for

A/E DOMAIN FACTOR NO. 1
CPU TIME CONSTRAINED (Continued)

ACTION CONSIDERATIONS: (Continued)

Factor No. 5 - Response Time, in Appendix A.

- Consider hardware tradeoffs to determine if a faster CPU is available, and if it would be a cost effective alternative.
- Consider relaxation of the response time requirements (user concurrence should be requested and documented).

CPU TIME CONSTRAINT	
EFFECT ON PRODUCTIVITY:	
COMMAND & CONTROL:	35% DECREASE
SCIENTIFIC:	40% DECREASE
UTILITY:	55% DECREASE
BUSINESS:	NO EFFECT
ALL OF THE ABOVE:	25% DECREASE

A/E DOMAIN FACTOR NO. 2
PROGRAM MEMORY SIZE CONSTRAINED

QUESTION: Is the program memory size of the processor a constraint to the software development?

GENERAL IMPACT: If the software development is constrained by the size of the processor program memory, then costs are expected to increase over what one would expect without the constraint.

ACTION CONSIDERATIONS:

- Determine size requirements of the program memory.
  - If the estimate of requirements is less than 60 percent of total memory, assume little or no effort.
  - If the estimate is greater than 60 percent and less than 80 percent of total memory, anticipate increased costs (15 to 20 percent as appropriate.)
  - If the estimate is greater than 80 percent, assume major impact on costs (an increase of as much as 200 percent can occur).
- If memory utilization is greater than 60 percent,
  - consider hardware and firmware trade-offs as discussed under Factor No. 5 - Response Time, of Appendix A. The least expensive hardware alternative is to add memory to the proposed Control Processing Unit (CPU). If, however, the proposed CPU is fully configured with memory, then this will not be a viable alternative,
  - consider relaxation of operational requirements to decrease memory requirements (user concurrence should be requested and documented),

A/E DOMAIN FACTOR NO. 2
PROGRAM MEMORY SIZE CONSTRAINED (Continued)

ACTION CONSIDERATIONS: (Continued)

- if required, accept a reduction in programmer productivity for the necessary extra effort required to make the software fit, with concomitant increase in software cost.

- Determine if additional memory or a larger CPU is available, and if it would be cost effective to implement the change(s).

PROGRAM MEMORY SIZE CONSTRAINT	
EFFECT ON PRODUCTIVITY:	
COMMAND & CONTROL:	20% DECREASE
SCIENTIFIC:	20% DECREASE
UTILITY:	15% DECREASE
BUSINESS:	UNDETERMINED
ALL OF THE ABOVE:	30% DECREASE

A/E DOMAIN FACTOR NO. 3
ON-LINE OPERATION

QUESTION: Does the program operate in the on-line or utility mode, such as scientific subroutines or code conversion routines?

GENERAL IMPACT: If the program operates in an on-line or utility mode in conjunction with the other significant effects, expect a decrease in costs of 70 percent in the portion of the software that affects this mode.

ACTION CONSIDERATIONS:

- Developing support software that operates in an on-line or utility mode, such as scientific subroutines, code conversion routines, and standard listing programs, is more productive in terms of cost per unit line of delivered code.
- If this type of software represents less than 10 percent of the total development, no special provisions are necessary.

ON-LINE OPERATION
EFFECT ON PRODUCTIVITY: INCREASE

A/E DOMAIN FACTOR NO. 4
TIME AND MEMORY CONSTRAINED

QUESTION: Is it projected that the CPU will be in both a time and memory constrained mode?

GENERAL IMPACT: If the software development is constrained in both the time and memory domains of the CPU, then costs are expected to increase by 150 percent over that expected with either constraint taken individually.

ACTION CONSIDERATIONS:

- Use the guidelines presented under Factor No. 1 - Time, and Factor No. 2 - Memory, of this Appendix, treated individually.
- This factor is very important for on-board flight programs in avionics where the combination of quick reaction real-time processing and weight and volume restrictions usually means both constraints are present.
- Determine if a faster and larger CPU is available and if it would be cost effective to implement a change.
- Determine if response time can be decreased and the software program reduced or modified.

TIME AND MEMORY CONSTRAINT
EFFECT ON PRODUCTIVITY: 60% DECREASE

AD-A044 609 DOTY ASSOCIATES INC ROCKVILLE MD F/G 9/2
SOFTWARE COST ESTIMATION STUDY. VOLUME II. GUIDELINES FOR IMPRO--ETC(U)
AUG 77 D L DOTY, P J NELSON, K R STEWART F30602-76-C-0182
UNCLASSIFIED TR-151-VOL-2 RADC-TR-77-220-VOL-2 NL

2 OF 2
AD A044609

END
DATE FILMED
10-77
DDC

A/E DOMAIN FACTOR NO. 5
TARGET CPU DESIGNATION

QUESTION: At what point in the schedule is the CPU or CPUs to be specified?

GENERAL IMPACT: The later in the schedule the CPU or CPUs are specified, the larger the impact on software development costs. The major impact, however, is on total development costs. The magnitude of the impact is not well known quantitatively, but it is considered significant.

ACTION CONSIDERATIONS:

- In many large weapon systems developments, software turns out to be on the critical path, since major efforts on the software cannot start until the source selection for hardware has been completed.
- Guidelines to cushion the impact of the time at which CPUs are specified in the schedule are as follows:
  - Specify, in the performance specification, which CPU is to be used.
  - Force the hardware contractor to select from standard military hardware, thereby greatly reducing software development uncertainty.
  - Develop as much software as possible in standard High Order Languages (HOLs), thereby greatly reducing hardware dependence.

LATE DESIGNATION OF CPU
EFFECT ON COST: VARIABLE

A/E DOMAIN FACTOR NO. 6
DESIGN STABILITY

QUESTION: How stable is the design?

GENERAL IMPACT: Instability in design can cause cost increases as large as 100 percent. If requirements change, causing design changes, see Factor No. 2 - Changes in Requirements, in Appendix A.

ACTION CONSIDERATIONS:

- Since 60 percent of the errors discovered in testing are usually caused by faulty design, some instability in design should be assumed.
- There is no iron-clad way of ensuring an initial stable design. The use of formal requirements analysis and design languages, as discussed in Factor No. 1 - Operational Requirements, in Appendix A, may tend to increase design stability.
- The use of modern programming may also increase design stability.
- For large projects, design changes are probably inevitable; therefore, leave some flexibility in both schedule and cost to account for this eventuality.
- Work closely with the user to insure that the initial design is as definitized as possible, and that the design is stabilized to the maximum extent to preclude subsequent design changes once coding has been commenced.

DESIGN STABILITY
EFFECT ON PRODUCTIVITY: 50% DECREASE OVER SPECTRUM FROM NO DESIGN CHANGES TO COMPLETE REDESIGN

A/E DOMAIN FACTOR NO. 7
DESIGN COMPLEXITY

QUESTION: How complex is the design, that is, how complicated and involved are the logic and software/hardware interfaces?

GENERAL IMPACT: Increased complexity decreases productivity, but no successful rating scales have been devised for measuring it.

GUIDELINES:

- The complexity of a project has a significant adverse effect on programmer productivity. General rules of thumb to keep in mind in this venue are:
  - Operating systems are more complex than compilers, and compilers are more complex than applications software. Support software, in general, is more complex than applications software.
  - Real-time applications are more complex than non-real-time applications.
  - Interrupt driven multi-tasking software is more complex than non-interrupt driven single tasking software.
- Complexity can be looked upon as an overview of a number of items covered by other factors. Once the design has been approved after Critical Design Review (CDR), then it will probably be of benefit for the project director to break up the software by levels of complexity, and cost each portion separately.

INCREASED COMPLEXITY
EFFECT ON PRODUCTIVITY: DECREASE

APPENDIX C

DISCUSSION OF MANAGEMENT DOMAIN FACTORS

FACTOR IMPACT CONVERSION

Refer to note on page A-1

MANAGEMENT DOMAIN FACTOR NO. 1
SUPPORT SOFTWARE AVAILABILITY

QUESTION: What sort of support software is available for the development?

GENERAL IMPACT: The availability and quality of support software has a large impact on development costs, but its magnitude is not easily quantifiable.

GUIDELINES:

- If either the development computer or the target computer or both are new, expect to pay a considerable amount for support software relative to operational software, compared to a development where these conditions do not exist.
  - If possible, try to avoid using a new computer as either the development or target machine. If a new computer is chosen, the advantages it provides should clearly outweigh the additional cost that will be required to develop adequate support software.
- With the larger main frames for the target computer, expect the quality and availability of support software to be better. This will be the case for command and control applications.
- With minicomputers and microprocessors for the target machine, expect the quality and availability of support software to get poorer. This will primarily be the case for avionics applications.

SUPPORT SOFTWARE AVAILABILITY
EFFECT ON COST: VARIABLE

MANAGEMENT DOMAIN FACTOR NO. 2
WORK BREAKDOWN STRUCTURE (WBS)

QUESTION: Is the Work Breakdown Structure (WBS) adequate for collecting software costs?

GENERAL IMPACT: It is possible that a Work Breakdown Structure (WBS) with only a single element for software will capture only 20 percent of the actual cost of developing software.

GUIDELINES:

- The structure of the Work Breakdown Structure (WBS) is critical in measuring the actual development cost of software for an embedded computer system. The WBS for a system with embedded computers will contain much more than elements related to software. Systems with embedded computers are the general rule for both command and control and avionics applications. Therefore, it is essential that the WBS be constructed properly to isolate actual software cost.
- Guidelines to ensure that software is reflected properly in the WBS are as follows:
  - A single element in the WBS for software will very seldom account for the total software development cost. Usually, a single element in the WBS for software will only account for coding and checkout costs, normally about 20 percent of total software effort. Software will not appear above level 3 in the WBS, if MIL-STD 881A is adhered to. Therefore, to account for software adequately, deeper levels of the WBS will be required.
  - It is imperative that analysis and design, and testing and integration be reflected in the WBS. Since the WBS for most systems with embedded computers will be oriented around prime mission equipment elements, the portions of each prime mission

MANAGEMENT DOMAIN FACTOR NO. 2
WORK BREAKDOWN STRUCTURE (WBS) (Continued)

GUIDELINES: (Continued)

equipment element targeted for software implementation should have separate software elements for analysis and design, coding and checkout, and testing and integration.

- Make sure that management and support costs for software development are adequately reflected in the WBS. In order to do this, it is usually at least necessary to put software elements in the System Engineering/Project Management portion of the WBS. These elements should be partitioned by the software life-cycle phases.
- Factoring hardware from software in a satisfactory manner in the WBS is very difficult for many developments. Many engineers, especially in avionics applications, are dually qualified in both hardware and software. Partitioning their time accurately among the various WBS elements is very difficult. This is especially difficult in the testing and integration phase, since the root of many problems encountered is not known as to hardware or software cause until they have been resolved. The only solution appears to be constant supervision so that labor costs are partitioned as accurately as possible between the hardware and software elements in the WBS.

WBS FRAMEWORK
EFFECT ON COSTS: VARIABLE

MANAGEMENT DOMAIN FACTOR NO. 3
DEGREE OF INNOVATION

QUESTION: How much innovation will be required in the development?

GENERAL IMPACT: This is essentially a catchall factor covered elsewhere. It includes special displays, concurrent development of other ADP components, a new development or target CPU, and new languages. The impact of these factors taken individually or in combination is large.

GUIDELINES:

- Anything that falls into the category of being new or innovative will have an adverse impact on software development costs.
- If existing hardware, techniques, or languages can be subsituted for new innovations, then the proposer of the innovations should show cause why the new innovations are required.
- Consider trade-offs to show the cost effectiveness of all innovations.

DEGREE OF INNOVATION
EFFECT ON PRODUCTIVITY: DECREASE

MANAGEMENT DOMAIN FACTOR NO. 4
TESTING REQUIREMENTS INCLUDING VERIFICATION AND VALIDATION

QUESTION: What testing requirements, including verification and validation, are to be imposed on the development?

GENERAL IMPACT: The imposition of specific testing requirements, such as Independent Verification and Validation (IV&V), can increase development costs by as much as 20 percent. However, because of the associated higher quality, these requirements should result in reduced maintenance costs.

GUIDELINES:

- Independent V&V should increase the quality of delivered software, but expect it to increase development costs by 20 percent. Independent V&V should probably not be a requirement for small projects, but should be given serious consideration for large projects.
- Testing requirements are usually specified in terms of some percentage of logic paths explored. The number of possible logic paths will increase geometrically with the size of developed code. Therefore, expect to explore a greater percentage of logic paths in a small development than a large development. Expect testing costs to be directly proportional to the percent of possible logic paths explored. Do not select the paths at random. Select on the basis of their expected frequency of use in an operational environment. Test those which are expected to occur most frequently.

MANAGEMENT DOMAIN FACTOR NO. 4
TESTING REQUIREMENTS INCLUDING VERIFICA- TION AND VALIDATION (Continued)

GUIDELINES: (Continued)

- Since the correction of errors discovered in testing reintroduces the probability of error (i.e., there is a 40 percent chance that correcting an error will reintroduce a new error),[21] regression testing requirements are sometimes imposed. This involves testing some percentage of the logic paths which are dependent on the path where the initial error was discovered. Follow the same guidance as above for primary paths.
- For on-board flight programs in avionics applications, expect to test a high percentage of possible logic paths, especially for software which is classified life critical. For simulation, expect the percentage to be lower, and for ATE, expect the percentage to be yet even lower.
- Testing requirements for command and control usually fall between on-board flight programs for avionics and simulation for avionics.

---

21. Barry W. Boehm, "Software Reliability and Measurement," TRW Corporation, Presentation given at Software Management Conference, Washington, D.C., March 22-23, 1976, sponsored by American Institute of Aeronautics and Astronautics.

TESTING REQUIREMENTS
EFFECT ON COST: INCREASE FOR TESTING & 20% INCREASE FOR IV&V

MANAGEMENT DOMAIN FACTOR NO. 5
COST/SCHEDULE CONTROL SYSTEMS CRITERIA (C/SCSC)

QUESTION: What is the Cost/Schedule Control Systems Criteria (C/SCSC) setup for the system?

GENERAL IMPACT: The impact of C/SCSC, or its non-formal equivalent, is directly dependent on the Work Breakdown Structure (WBS) for the development. See Factor No. 2 - WBS, in this Appendix, for impact.

GUIDELINES:

- The C/SCSC, or its non-formal equivalent, is the principal mechanism for determining if the project is deviating from planned cost and schedule. The reporting vehicle for C/SCSC is the Cost Performance Report (CPR). C/SCSC will only be as effective as the WBS for the development. If only one software element is in the Work Breakdown Structure (WBS), then C/SCSC will not be effective for software control.
- If the CPR, or its non-formal equivalent, contains more than one software element, then constant surveillance of cost and schedule variance should be maintained. There are a number of techniques for analyzing cost and schedule variance to affect project control which should be implemented.[22]

---

22. "Analysis of Measurement Data for the Surveillance of Cost/Schedule Control Systems Course (SVS 361)," Department of Management Techniques, School of Systems and Logistics, Air Force Institute of Technology, Wright-Patterson AFB, August 1975.

MANAGEMENT DOMAIN FACTOR NO. 5
COST/SCHEDULE CONTROL SYSTEMS CRITERIA (C/SCSC) (Continued)

GUIDELINES: (Continued)

- A simplified rule of thumb can be used as another cost control mechanism. It is the 40-20-40 rule: 40 percent of development effort in analysis and design, 20 percent in coding and checkout, and 40 percent in testing and integration. This, however, may be altered somewhat with the imposition of modern programming techniques. The first of these development phases has a milestone usually associated with it, the Critical Design Review (CDR). If the CDR, which signals the end of design, has not been completed by the time 40 percent of the funds has been expended, then one should be aware that there may be a potential cost overrun in the offing.

C/SCSC
EFFECT ON COST: VARIABLE

MANAGEMENT DOMAIN FACTOR NO. 6
DEVELOPMENT PERSONNEL MIX

QUESTION: What is the mixture of support personnel to programmers and analysts on the development?

GENERAL IMPACT: Each 10 percent increase in support personnel relative to programmers and analysts will increase cost per unit line of delivered code by 25 percent.

GUIDELINES:

- The expected mix of support personnel (management, clerical, etc.) to programmers and analysts is 20 percent support personnel to 80 percent programmers/analysts. Deviations from this mix depend on project peculiarities. Expect the impact indicated above when this occurs.
- If the developer has a mix sharply different from the expected, have the developer show justification for the mix.

PERSONNEL MIX
EFFECT ON PRODUCTIVITY: 25% DECREASE FOR EACH 10% INCREASE IN SUPPORT PERSONNEL

MANAGEMENT DOMAIN FACTOR NO. 7
PROGRAMMER TESTING

QUESTION: Are programmers given hands-on computer availability for their own testing?

GENERAL IMPACT: Submitting programs to be tested and run by a separate computer operations staff is about 50 percent more productive than giving programmers hands-on computer availability.

GUIDELINES:

- This factor only applies to batch environments on large main frames. It does not apply to time-sharing or where the development Central Processing Unit (CPU) is either a mini- or microcomputer.
- If the development CPU is a large main frame operating in a batch environment, then:
  - a developer who has a separate computer operations staff and limits programmer hands on CPU availability should be put in a more favorable light than one who does not;
  - attempt to keep the programmers confined to programming and preparing test runs, and let the computer operations staff make the runs on the computer;
  - if a large percentage of machine checkout and testing is done by programmers instead of the computer operations staff, then expect lower programmer productivity and a concomitant higher cost.
- There is a fine balance between bench checking and machine testing programs in terms of achieving optimum productivity. Two runs per day in a batch environment seem to be about optimum for machine testing.

MANAGEMENT DOMAIN FACTOR NO. 7
PROGRAMMER TESTING (Continued)

GUIDELINES: (Continued)

- This factor is of small importance in avionics since it is usually a minicomputer or microcomputer environment.
- For command and control applications, this factor is very likely to arise.

PROGRAMMER TESTING -- BATCH ENVIRONMENT ON LARGE FRAMES
EFFECT ON PRODUCTIVITY: LIMITED COMPUTER ACCESS INCREASES PRODUCTIVITY 50%

MANAGEMENT DOMAIN FACTOR NO. 8
AMOUNT & METHOD OF COST DATA COLLECTION

QUESTION: What is the amount and method of cost data collection?

GENERAL IMPACT: Increasing the amount of cost data collected will increase development cost. Manual collection of data is more costly than methods which depend on automated means. Costs associated with data collection may be recouped in a smoother running project. If an existing cost collection system is to be used, then there is no impact on cost.

GUIDELINES:

- If the existing cost collection system, in place at the developer, is considered adequate, data collection costs should be minimal. Contractors which lack an acceptable in-place cost collection system should be required to implement one, and accordingly may require additional funding.
- The amount of data to be collected is a direct function of the Work Breakdown Structure (WBS) and the Cost/Schedule Control Systems Criteria (C/SCSC) setup for the development.
- If additional data is requested that is not part of the developer's ordinary cost data collection practices, then the following should be considered:
  - Determine if the expected increase in development costs to implement new cost collection practices can be recouped during the current development, assuming the implementation is not required. If not, determine if the new practices implemented will be beneficial to other subsequent developments.

MANAGEMENT DOMAIN FACTOR NO. 8
AMOUNT & METHOD OF COST DATA COLLECTION (Continued)

GUIDELINES: (Continued)

- Assess if the new cost collection practices should be implemented by manual or automatic means. Manual implementation will cost less to develop than automatic, but will carry a higher operational cost. The trade-off should be examined carefully, especially its carryover to subsequent developments. Some cost collection practices will not be easily amenable to implementation by automatic means.

- Serious consideration should be given to the collection of product information, such as instruction count, as well as cost information. It helps to have something to measure cost against in terms of product standards and status.

COST DATA COLLECTION
EFFECT: VARIABLE

MANAGEMENT DOMAIN FACTOR NO. 9
COST OF SECONDARY RESOURCES

QUESTION: What secondary resources are being used in the development?

GENERAL IMPACT: Secondary resources on the average amount to about 7.5 percent of total development costs. They include primarily computer time and documentation production costs.

GUIDELINES:

- On the average, expect about 4 to 5 hours of computer time per man-month in a batch single partition environment. Costs will be equivalent, but time will not be the same, in terminal oriented or multi-programming environments.
- On the average, expect about 5 pages of documentation per man-month.
- If secondary resources are to be accounted for accurately, then the WBS has to be structured properly to account for them.
- Any algorithm used to estimate secondary resources should be consistent with the WBS. For example, if the algorithm estimates only computer time and documentation production cost, then the WBS should have specific elements for these items. Often, secondary resources will be buried in overhead or other non-software elements in the WBS. In these cases, it will be virtually impossible to measure the actual cost of secondary resources. Therefore, any algorithm used to estimate them will prove of little use, since there will be little to compare them to.

SECONDARY RESOURCES
EFFECT ON COST: 7.5% OF PRIMARY RESOURCE COST

MANAGEMENT DOMAIN FACTOR NO. 10
DEFINITION OF INSTRUCTION

QUESTION: If instruction count is being used as the sizing parameter for cost estimation purposes, which of the many definitions of the word "instruction" is being used?

GENERAL IMPACT: Worst case is 300 percent (the expansion ratio) error when using object words in a Cost Estimating Relationship (CER) developed on the basis of source statements, or vice versa. Other additive errors can occur in treating delivered vs. non-delivered code, support software, the handling of comment and copy statements, etc.

GUIDELINES:

- To estimate costs reliably, the definition on which the instruction count is based must be consistent with that used in developing the Cost Estimating Relationship (CER).
  - If the CER was developed on the basis of object code, then;
    - instruction count should be in object code,
    - handle data areas and constants consistent with the CER,
    - handle reusable code consistent with the CER,
    - handle deliverable vs. non-deliverable code consistent with the CER, and,
    - handle support vs. operational software consistent with the CER.
  - If the CER was developed on the basis of source code, then;
    - instruction count should be in source statements,
    - handle comment, copy, and declarative statements consistent with the CER,

MANAGEMENT DOMAIN FACTOR NO. 10
DEFINITION OF INSTRUCTION (Continued)

GUIDELINES: (Continued)

- handle reusable code consistent with the CER,
- handle deliverable vs. non-deliverable code consistent with the CER, and,
- handle support vs. operational software consistent with the CER.

DEFINITION OF INSTRUCTION
EFFECT ON COST: VARIABLE -- WORST CASE = 300% ERROR

MANAGEMENT DOMAIN FACTOR NO. 11
SIZING ERROR

QUESTION: What is the effect of a sizing error on the cost estimate?

GENERAL IMPACT: If the cost per instruction method is used for the cost estimate where size is the number of instructions, then the effect of a sizing error has a direct impact on the error in the cost estimate. Since sizing estimates can be off by as much as 200 percent, an error of greater than 200 percent can be injected into the cost estimate.

GUIDELINES:

- Sizing error will get smaller as the project moves toward completion.
- Since the error associated with programmer productivity changes as a function of the instruction count, the appropriate sizing parameter should be selected as a function of where this program is in its development. Use the following guidelines for selection:
  - Conceptual Phase - Initial Budgetary Estimate.
    - Total size in object words (greater than 200% error).
  - Validation prior to release of RFP.
    - Size in object words minus data areas (greater than 100% error)
  - After receipt of proposals through PDR.
    - Size in new object words minus data areas after adjustment for reusable code (greater than 75% error).
  - From PDR through remainder of development.
    - Size in new source statements (initial 50% error improving to 0% at completion).

SIZING ERROR
EFFECT ON COST: VARIABLE

MANAGEMENT DOMAIN FACTOR NO. 12
DATA MANAGEMENT TECHNIQUES

QUESTION: What kind of computer data management techniques are to be used for the development?

GENERAL IMPACT: The impact of using a Data Base Management System (DBMS) versus a File Management System for computer data handling is not known quantitatively, but it is expected to be significant.

GUIDELINES:

- A File Management System will reduce development costs, but increase maintenance costs.
- A DBMS will increase development costs, but decrease maintenance costs.
- If there is not expected to be much volatility in the types and format of data to be handled over the life of the system, then opt for a File Management System.
- If great volatility is expected, then opt for a DBMS.
- Expect to pay an efficiency penalty in both the CPU time and memory domains when using a DBMS.
- If the choice is left in the hands of the developer, the project manager should be aware of the trade-off between development and maintenance costs.
- This is not a factor of importance for avionics applications, since any data management that will be required can usually be handled easily by a simple File Management System.

MANAGEMENT DOMAIN FACTOR NO. 12
DATA MANAGEMENT TECH- NIQUES (Continued)

GUIDELINES: (Continued)

- For command and control applications, this factor can be important, especially for a system with large data management tasks.

DATA MANAGEMENT TECHNIQUES
EFFECT ON COST: VARIABLE

MANAGEMENT DOMAIN FACTOR NO. 13
MODERN PROGRAMMING TECHNIQUES

QUESTION: Are modern programming techniques to be used for the development?

GENERAL IMPACT: Structured top-down design along with all the associated disciplines can decrease cost by up to 40 percent over the same development using non-structured, non-top-down design methods.

GUIDELINES:

- The maximum benefit from modern programming techniques will be realized from large programs, in general 100,000 lines of source code and greater.
- The benefit derived from the use of modern programming techniques will be a function of the number of associated disciplines implemented, such as Chief Programmer Teams, Programming Support Libraries, and Hierarchy Input Process Output (HIPO). The imposition of some of these disciplines may involve additional investment costs. For example, the development computer proposed may not provide Programming Support Libraries. In that case, an investment would have to be made in support software development to provide Programming Support Libraries for the development computer.

MODERN PROGRAMMING
EFFECT ON PRODUCTIVITY: 67% INCREASE

MANAGEMENT DOMAIN FACTOR NO. 14
PROGRAMMING FACILITIES

QUESTION: What sort of programming facilities are available for the development?

GENERAL IMPACT: The quality and availability of programming facilities, such as computer facilities, support software, and personnel, have a large impact on development costs, but the magnitude is not easily quantifiable.

GUIDELINES:

- Let the developer control the programming facilities to as high a degree as possible. This includes:
  - development at the developer's site instead of a purchaser selected site, (e.g., operational site) and,
  - development on a developer controlled dedicated computer instead of a computer run by another organization.
- There may be extenuating circumstances where this is impractical. For example, the cost of supplying the developer with the development CPU may be large compared to making time available to the developer on a non-developer controlled computer.

PROGRAMMING FACILITIES
EFFECT ON PRODUCTIVITY: VARIABLE

MANAGEMENT DOMAIN FACTOR NO. 15
DEVELOPMENT AND TARGET COMPUTER DIFFERENT

QUESTION: Is the target computer different than the computer on which the software is to be developed?

GENERAL IMPACT: If this occurs along with other significant effects, the costs are expected to increase depending on the program application.

GUIDELINES:

- If the target computer is the same as that on which development is to be performed, no effect is anticipated.
- If computers are different, the following steps should be taken:
  - If adequate support software is not available for the target computer, then it is better to utilize the development computer proposed.
  - If adequate support software is available for the target computer, have the developer show cause why the software is not being developed on the target computer.
- If coded on a large computer, expect slightly more efficient coding (on smaller computers, expect less efficient coding). This is a potential reason for having the development and target computer different.
- Be prepared to accept increased costs and schedule slippages (including the development of support software for the target computer).

MANAGEMENT DOMAIN FACTOR NO. 15
DEVELOPMENT AND TARGET COMPUTER DIFFERENT (Continued)

GUIDELINES: (Continued)

- Regarding the target computer, as one moves toward larger, more powerful main frames, the likelihood of adequate support software increases, thus increasing the likelihood that the target and development computer will be the same. This is primarily the case for command and control applications. As one moves toward minicomputers and microprocessors, the likelihood of adequate support software decreases, thus increasing the likelihood that the target and development computer will be different. This is primarily the case for avionics applications.

DEVELOPMENT AND TARGET TARGET COMPUTER DIFFERENT	
EFFECT ON PRODUCTIVITY:	
COMMAND & CONTROL:	55% DECREASE
SCIENTIFIC:	10% DECREASE
UTILITY:	30% DECREASE
BUSINESS:	NO EFFECT
ALL OF THE ABOVE:	20% DECREASE

MANAGEMENT DOMAIN FACTOR NO. 16
COMMUNICATIONS

QUESTION: What degree of effective communications have been established between the developer, purchaser, maintainer, and end user?

GENERAL IMPACT: The impact of this factor is not known quantitatively, but it is expected to be considerable.

GUIDELINES:

- In some developments, the developer, purchaser, maintainer, and end user are all separate and distinct parties. In others, two or more of the functions may be combined in a single party. For example, the purchaser and end user may be the same party. For software developed by the Air Force Systems Command, each function is performed by a separate party. The developer is usually a contractor, the purchaser is a SPO residing in the Systems Command, the maintainer is the Air Force Logistics Command, and the end user is an operational command, such as SAC.
- The developer has to satisfy the requirements of the purchaser, maintainer, and end user. It is usually the purchaser's responsibility that the maintainer's and end user's requirements get communicated to the developer. Do not permit direct contact between the maintainer or end user and the developer. The Air Force has set up guidelines and procedures to insure

MANAGEMENT DOMAIN FACTOR NO. 16
COMMUNICATIONS (Continued)

GUIDELINES: (Continued)

that the end user's and maintainer's requirements are communicated through the purchaser to the developer, in accordance with AFSCP 800-3. These guidelines and procedures should remain high on the SPO's awareness scale because of their importance to effective software development.

INEFFECTIVE COMMUNICATION
EFFECT ON COST: INCREASE

MANAGEMENT DOMAIN FACTOR NO. 17
LANGUAGE REQUIREMENTS

QUESTION: What language or languages are to be used in the development?

GENERAL IMPACT: A complete Machine Oriented Language (MOL) development can cost up to 400 percent more than an entire High Order Language (HOL) development.

GUIDELINES:

- The percentage of High Order Language (HOL) versus Machine Oriented Language (MOL) should not be an edict at the outset. If possible, simply specify that 100 percent of the development should be in HOL. For certain types of software, HOL generates intolerably inefficient object code in both the time and memory domains on the target CPU. When this is the case, MOL is the only choice. With this consideration in mind, the following steps should be taken:
  - For each function to be performed, assess the efficiency requirement. If the particular task can be performed efficiently by an HOL, select the HOL for that function. If not, assign it for an MOL implementation. For the larger more powerful main frames, the necessity for MOL implementation is less, as is the case for command and control applications. For minicomputer and microprocessor implementation, the necessity for MOL increases; primarily in the case of on-board flight programs in avionics applications.

MANAGEMENT DOMAIN FACTOR NO. 17
LANGUAGE REQUIREMENTS (Continued)

GUIDELINES: (Continued)

- Once the functions in the development have been categorized into HOL and MOL implementation, the amount of source code required and the resultant development cost can be determined.

- For command and control applications, expect a high degree of HOL implementation. For on-board flight programs in avionics applications, expect a low degree of HOL implementation at present but expect it to increase in the future. For simulation and ATE in avionics applications, expect a higher degree of HOL implementation, but not as high as for command and control applications.

LANGUAGE REQUIREMENTS
EFFECT ON COST: ALL MOL UP TO 400% MORE THAN ALL HOL

MANAGEMENT DOMAIN FACTOR NO. 18
DEVELOPMENT SITE

QUESTION: Is the software development to be performed at the developer's facility or an operational site?

GENERAL IMPACT: If the software is to be developed at the developer's facility, the cost could be expected to be 45 percent less than if developed at an operational site.

GUIDELINES:

- If software is to be developed at the developer's facility, rather than at an operational site, anticipate lower costs.
- If the software is to be developed at an operational site (government facility), re-evaluate the requirements for this because of the associated increase in cost (e.g., security requirements, SPO required on-site interface, joint development, etc.).
- Perform trade-offs of cost and benefits of alternative sites. Benefits could involve the development and target computer being the same, and mitigation of CPU time and memory constraints which could occur on developer's processor.

DEVELOPMENT AT OPERATIONAL SITE
EFFECT ON PRODUCTIVITY: 30% DECREASE

MANAGEMENT DOMAIN FACTOR NO. 19
DEVELOPER USING ANOTHER ACTIVITY'S COMPUTER

QUESTION: Is the computer, upon which the software is being developed, operated by an activity other than the software development activity?

GENERAL IMPACT: If the development computer is operated by another activity, the cost is expected to increase by 45 percent.

GUIDELINES:

- If the software developer is using a computer at another activity, e.g., at a government facility, for the software development, anticipate lower programmer productivity (higher costs).
- If software development is being performed on a computer at the developer's facility, anticipate no impact on costs or schedule.

DEVELOPMENT COMPUTER
EFFECT ON PRODUCTIVITY: 30% DECREASE IF AT OTHER ACTIVITIES

MANAGEMENT DOMAIN FACTOR NO. 20
NUMBER OF DEVELOPMENT LOCATIONS

QUESTION: Is the software being developed at more than one site (location)?

GENERAL IMPACT: If the software is being developed at more than one site, costs are expected to increase by 25 percent.

GUIDELINES:

- If the software is being proposed to be developed at more than one site, then the following should be considered:
  - Have the developer justify multi-site development. Peculiar end-user requirements where each installation has site dependent software, and the developer has to be on-site is a possible justification. The internal corporate structure of the developer as a reason will be harder to justify.
  - A developer who proposes a single site development should have an advantage over a developer who has his proposed software team spread over different locations.
  - Examine the feasibility of having the developer bring his entire proposed software team together at one site.
  - For large systems, the likelihood of multi-site development increases. In many cases, avoiding it may be impossible.

MANAGEMENT DOMAIN FACTOR NO. 20
NUMBER OF DEVELOPMENT LOCATIONS (Continued)

GUIDELINES: (Continued)

- If multi-site development occurs, expect increased costs and some slippage in schedule.

MULTISITE DEVELOPMENT
EFFECT ON PRODUCTIVITY: 20% DECREASE

MANAGEMENT DOMAIN FACTOR NO. 21
CONCURRENT DEVELOPMENT OF HARDWARE

QUESTION: Is the software being developed concurrent with the hardware?

GENERAL IMPACT: If there are concurrent ADP hardware and software developments, costs are expected to increase depending on the program application.

GUIDELINES:

- If the software is being developed concurrently with the hardware, determine what percent of the software is affected by this concurrent hardware development.
  - If less than 10 percent of software is affected by the hardware development (90 percent of software can be developed without effect), the effect will be minimal.
  - If greater than 10 percent, expect increased costs and maintain a management reserve of funds.
- Determine if off-the-shelf hardware can directly perform the function of the developmental hardware.
  - If yes, assess resultant software and hardware cost impacts, as appropriate.
  - If no, determine whether off-the-shelf hardware can be adapted or modified for use. (Assess the cost impact of hardware and software modifications, as appropriate).

MANAGEMENT DOMAIN FACTOR NO. 21
CONCURRENT DEVELOPMENT OF HARDWARE (Continued)

GUIDELINES: (Continued)

- If no, also determine the percent of software affected by available hardware.

CONCURRENT DEVELOPMENT OF ADP HARDWARE	
EFFECT ON PRODUCTIVITY:	
COMMAND & CONTROL:	40% DECREASE
SCIENTIFIC:	55% DECREASE
UTILITY:	20% DECREASE
BUSINESS:	25% DECREASE
ALL OF THE ABOVE:	45% DECREASE

MANAGEMENT DOMAIN FACTOR NO. 22
DEVELOPER'S FIRST TIME ON SPECIFIED COMPUTER

QUESTION: Is this the first time the developer has used this computer to develop a software program?

GENERAL IMPACT: If this is the first time the developer has used the computer, the cost is expected to increase by 100 percent, and programmer productivity is expected to decrease 50 percent.

GUIDELINES:

- If this is the first time the developer has used this computer to develop a software program, anticipate a slippage in schedule and increased costs.
- If this is the first time the developer has used this computer to develop a software program, consider having developer use a computer with which he is familiar, but also consider the impact of Factor No. 15 - Development and Target Computer Different, in this appendix.

FIRST TIME ON COMPUTER
EFFECT ON PRODUCTIVITY: 50% DECREASE

MANAGEMENT DOMAIN FACTOR NO. 23
SPECIAL DISPLAY REQUIREMENTS

QUESTION: Is there a requirement for special displays?

GENERAL IMPACT: If the portion of the total software requirements, as represented by special displays, is more than 10 percent, the cost is expected to increase depending on the program application.

GUIDELINES:

- What portion of the total software requirements is required for special displays?
  - If the software requirements for special displays are less than 10 percent of the overall software requirements, the effect on costs and schedule should be negligible.
  - If the software requirements for special displays are greater than 10 percent, consider utilization of existing displays (with currently available supporting software) to reduce the requirements for special displays.
- Consider possible reduction and/or elimination of display complexity, if feasible, without seriously degrading overall operational performance.
- Expect a schedule slippage and increased costs.

SPECIAL DISPLAY	
EFFECT ON PRODUCTIVITY:	
COMMAND & CONTROL:	10% DECREASE
SCIENTIFIC:	10% DECREASE
UTILTIY:	NO EFFECT
BUSINESS:	30% DECREASE
ALL OF THE ABOVE:	10% DECREASE

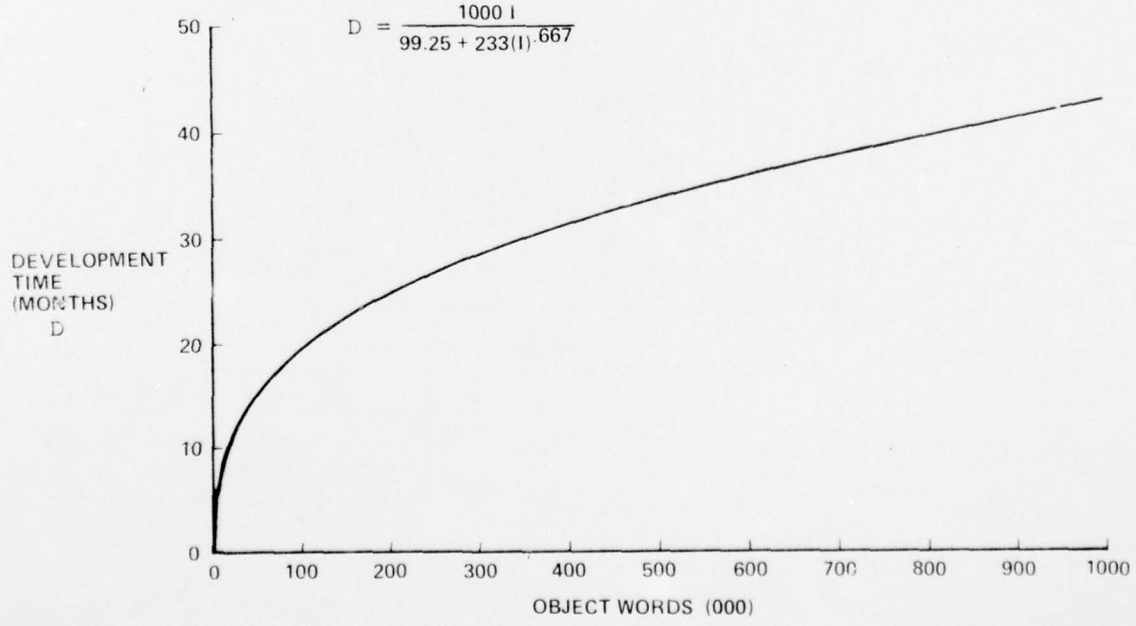
MANAGEMENT DOMAIN FACTOR NO. 24
SOFTWARE DEVELOPMENT SCHEDULE

QUESTION: What is the planned schedule for the proposed development, and what is the planned distribution of cost over the schedule?

GENERAL IMPACT: The impact of deviation from optimum schedule and resource distribution is not known exactly, but it is expected to be considerable.

GUIDELINES:

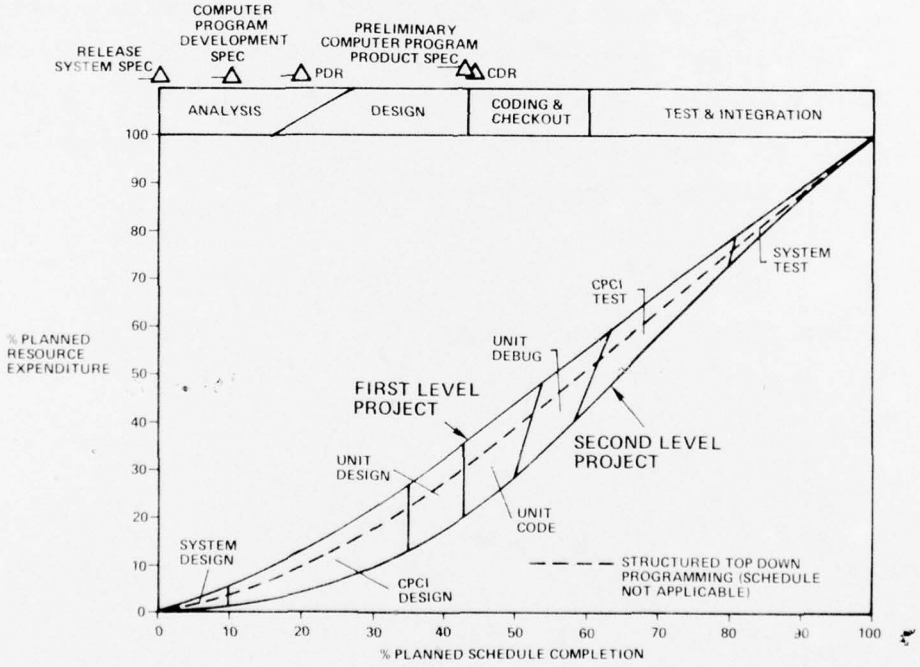
- The actual schedule (development time) to complete a software project is highly correlated with the size as measured by number of instructions.
- Use the curve or equation below to estimate the expected development time as a function of size.



MANAGEMENT DOMAIN FACTOR NO. 24
SOFTWARE DEVELOPMENT SCHEDULE (Continued)

GUIDELINES: (Continued)

- Expect some cost impact for deviations from the expected schedule.
- Expect the distribution of resources over the schedule to follow the approximate shape of the curve below.
- Expect some cost impact for deviations from the expected distribution of resources.



DEVELOPMENT SCHEDULE
EFFECT: SEE CURVES ABOVE

APPENDIX D
SOFTWARE SIZING METHODOLOGY

This appendix describes and demonstrates methods currently used to estimate the size of software programs. They are not sophisticated, nor are they particularly accurate. In addition, analytical models are presented which can be used, in lieu of better guidance, to estimate the size of software. The models are not considered good because their statistical performance parameters are low. In addition, several variables contained in the estimators are descriptors of the processor and its associated data base, information not always available in early phases of software development.

Relationships for estimating the resource requirements of software development invariably have program size, in object or source code, as an independent variable. But estimating the size of software programs has proven to be the most difficult aspect of, and the source of greatest error in, analyses to project resource requirements of software development.

There are several reasons for this occurring. Inadequate emphasis has been given to the development of techniques for estimating program size. Consequently, data has not been collected which would support the development of these methods. This dearth of data prevents the development of analytical techniques, as well as inhibits the ability of the software development community to draw analogies from prior developments.

Another source of error is inadequate preparation on the part of the developers to support size projections. Too often, size projections are made with little or no design analysis, the detailed study of what the software is to do and how it will do it. Experience has shown that software developers can underestimate size by a factor of three if insufficient attention is given to analyzing the software requirements. If development contracts

are awarded on the basis of cursory projections, drastic cost and time overruns will occur. Design analysis significantly improves the accuracy of software size estimation. And, to ensure that adequate consideration is given to the design requirements, the developer, as a minimum, should be required to provide a software Work Breakdown Structure (WBS), a functional flow diagram, and estimates of software sizes for each work package prior to initiation of the development. If possible, algorithms to be programmed should also be provided by the developer; this is a method of insuring that adequate thought has been given to the complexity of the problem.

In estimating software sizes by drawing analogy to prior development, care must be taken to ensure that the programs from which analogies are being drawn perform similar functions in a similar manner as the proposed programs. For example, in software developed by a contractor for several systems of the same type, the functions performed by software modules and routines, although identified by the same functional title, were different. To draw analogies accurately for proposed new systems of this type, it was necessary to review the software at the subroutine level.

Another source of sizing error, although not considered appreciable, is the different capabilities of the programmers which result in various degrees of code efficiency. If a developer is writing code in a memory constraint environment, his code efficiency will tend to be maximal. On the other hand, in a non-constrained environment, less experienced personnel may be utilized, and less efficient code and larger programs can result.

Thus, the potential causes of error in estimating software size are numerous, and they can be controlled best by allocating adequate time and resources to system requirements analysis.

D.1 System requirements analysis

If the software is being developed as part of the development of a hardware system, the system requirements analysis will be performed in a slightly different manner.

In conjunction with hardware system development, the system configuration is defined as distinct units with identified interfaces, of which any unit or system may contain computer processing or control. Requirements must be specified for each unit, with interfaces and relationships among the units clearly defined. Then, the system is analyzed functionally, with the functions performed by each unit delineated. This results in a functional block diagram and a listing of all functions performed by the system. From this, the functions performed by the software are identified, and are defined from the functional and requirements analysis.

As part of a software development program, say for command and control, in which software performs most, if not all, functions, the requirements analysis would be restricted to defining the functions and interfaces of the software modules.

The projected software sizes are thus based on the more comprehensive and definitive data evolving from these analyses.

D.2 Alternative approaches to software sizing

There are two methods used for sizing software early in the conceptual phase of development. The specific approach used is dictated by the experience of the estimator, and the extent and relevance of available historical data. Each of the approaches assumes that the mission analysis is complete, the operational requirements and major configuration constraints are identified, and that the operational concept, including its support concept, is understood. The methods entail the following:

- Specified computer hardware. The software sizing estimate is achieved by summing the core memory of the specified computer hardware, and adjusting the sum for assumed core overlays and expected core utilization.
- Software analogy. The software sizing estimate is derived by partitioning the software down to the functional subroutine level, and then estimating the size of each work package by analogy with similar programs/subroutines/work packages in past development projects.

Sizing software by analogy, which requires the availability of relevant historical data, is considered the more accurate procedure. However, the success of the approach is highly dependent on the accuracy and relevancy of the data from which the analogy is being drawn. Software development activities are becoming increasingly aware of the need for such historical data, and steps are being taken to assure its collection. If sufficient data exists, it is recommended that more than one approach be utilized and the results of the estimates compared.

Software sizing in the Conceptual Phase is made in terms of total object code. This is because object code can be estimated early in the development with greater accuracy than source code.* If both source and object code sizes were known, then one would choose source code because the error would be less.

Like the initial conceptual estimate, most subsequent estimates to PDR are made in object code. At that time there is sufficient information available to begin estimating in source code. Updates are made to the initial estimate by subtracting out data areas and reusable code as these factors become known. A final update takes place once the target computer and language mix are finalized. This is the point where the estimate is made in source code. Table D-1 summarizes the points in the development cycle where software sizing estimates are usually required.

---

*Use of source code at this early conceptual stage requires the estimator to make premature subjective judgments relative to the target computer, language mix, amount of reusable code, etc.

TABLE D-1. SOFTWARE SIZING ESTIMATING ERRORS

Software estimate	When	Sizing basis	% Error
1. Initial program budgetary estimate	Conceptual phase	Total object code	up to 200%*
2. Independent program validation cost estimate	Validation prior to RFP release	Total object minus data areas (Executable Code)	up to 100%
3. Independent FSD cost estimate	Completion of system Spec through PDR	Total object minus data areas with adjustments for reusable code	up to 75%
4. Update of FSD cost estimate	PDR through remainder of development	Total source code	up to 50%, improving to zero at completion

*The actual may be 200 percent of the estimated or the estimated may be 200 percent of the actual.

Estimating code in source lines can be accomplished with some confidence given the successful completion of the Preliminary Design Review (PDR). At this point the target computer is known as well as the language mix and the extent to which reusable codes will be available. The algorithm for converting from object code (I_o) to source code (I_s) is shown in the following illustrative example:

$$I_s = \frac{I_o}{1 + P_H (E_H - 1)} = \frac{250,000}{1 + .167(4-1)} = 166,670 \text{ instructions} \quad (1)$$

where I_o = 250,000 object instructions

P_H = .167, the fractional amount of HOL

E_H = 4, the expansion ratio for HOL.

D.2.1 Analysis of size by estimating core requirements. The following relationship, although derived from data with deficiences, can be used to estimate memory size requirements.

$$M = 0.177\ k \left[\frac{N_F^{0.337}\ W_s^{0.147}}{t_c^{0.770}} \right]$$

$(R^2 = .52,\ SE = .086\ \ln\ units)$

where

M = Memory size, in thousands of words of object code

N_F = Number of major functions to be performed by the software

W_s = *Word size, in bits*

t_c = Cycle time of processor, in microseconds*

k = A constant dependent on application = 2.573 for signal processing

2.727 for *missile fire* control

2.781 for interfacing

3.412 for communication

3.565 for navigation

4.046 for command and control

4.451 for weapon fire control

Perhaps the variable N_F, number of major functions, is defined best by examples--target tracking, target identification, navigation, system monitoring, display, steering, parameter measurement, tuning, *target data entry*,

*The time to either retrieve or store a word in the processor memory.

firing sequence control, etc. The variables t_c and W_s are essentially dictated by the CPU in the application. Based on the type applications, most of which are in real-time, few overlays would be expected. If, as in most cases, the core utilization can be assumed to be approximately 80 percent, the expression represents an estimator for software size.

D.2.2 Sizing by analogy. It is generally recommended that estimates of software size be derived through extrapolation from previously developed software. If the application is of the same type, e.g., command and control, and the functions of the system appear to be the same, direct projections with little adjustment for changes can be made. If the new application appears more complex (requires additional software functions), comparisons of the numbers and types of software functions will permit adjustments to be made to the words per function and to the number of functions.

There is evidence that within software modules, the sizes of major functions of the module appear to decrease cumulatively at a nominal 80 percent slope.* That is, if the number of functions doubles, the average size per function decreases 20 percent. If the program size and the number of major functions performed by software can be determined from analogous data, estimates can be made as to the size of new software as follows:

$$I_2 = \left[\frac{I_1}{N_{f_1}^{.678}}\right] N_{f_2}^{.678}$$

*Based on limited data in an analytical study of software packages for electronic equipment by Doty Associates, Inc. (DAI), in 1977 under Navy Contract No. N00039-76-C-0320.

where

I_1 is the size of the analogous software, in object words

I_2 is the size of the proposed program, in object words

N_{f_1} is the number of major functions performed by the analogous software

N_{f_2} is the number of major functions to be performed by the new software

D.2.3 Other models. Estimators of software size were developed from data available in the literature.[23] Variables considered as potential determinants of program size were selected from the data and the following relationships were derived using multivariate regression:

- $$I = .449 \left[\frac{W_d^{0.018} N_c^{0.360}}{t_a^{0.173} M_o^{0.105}} \right] \prod_{j=1}^{j=6} f_j$$

(R^2 = .45, SE = 1.26 ln units)

Factor		Yes	No
Special display	f_1	= 0.550	1.00
On-line application	f_2	= 1.420	1.00
Software interfaces	f_3	= 1.453	1.00
Application involves more than one ADP Center	f_4	= 1.947	1.00
Innovation required	f_5	= 2.271	1.00
Real-time application	f_6	= 6.913	1.00

23. Kossiakoff, A., et al. "DoD Weapon Systems Software Management Study," Johns Hopkins University Applied Physics Laboratory, June 1975.

- $$I = .017 \left[\frac{W_d^{0.005} N_c^{0.334} t_a^{0.117} c_s^{0.379} W_s^{0.373}}{M_o^{0.145}} \right] \prod_{j=1}^{j=6} f_j$$

(R^2 = .45, SE = 1.26 ln units)

Factor	Yes	No
Special display	f_1 = 0.498	1.00
On-line application	f_2 = 1.316	1.00
Software interfaces	f_3 = 1.658	1.00
Application involves more than one ADP Center	f_4 = 1.885	1.00
Innovation required	f_5 = 2.163	1.00
Real-time application	f_6 = 7.217	1.00

where

I = Program size, in thousands of lines of source code

W_d = Size of data base, in thousands of words

N_c = Number of classes of items in data base

t_a = Add time of processor, in microseconds

W_s = Size of memory word, in bits

c_s = Core size, in thousands of words

M_o = Number of message output types

The consistency in the values for the state variables, f_j, and in the exponents of the variables (except for t_a) in the two relationships reflects the relative insensitivity of these variables to the addition or removal of the other variables.

Unfortunately, the variables c_s, W_s, and t_a are dictated by the processor used, and values for W_d and N_c are not likely to be available early in the software design. Also, the variables in the relationships are poor descriptors of the variance in size (R^2 is low), and adding variables has no apparent effect on the R^2 or the standard error. Therefore, the relationships are not considered satisfactory for use in sizing software, unless no other guidance is available.

APPENDIX E

GLOSSARY OF ACRONYMS

ADP	Automatic Data Processing
APP	Advance Procurement Plan
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CPCI	Computer Program Configuration Item
CPU	Central Processing Unit
CRISP	Computer Resources Integrated Support Plan
C/SCSC	Cost/Schedule Control Systems Criteria
DCP	Decision Coordination Paper
DID	Data Item Description
DSARC	Defense Systems Acquisition Review Council
DTC	Design to Cost
ECP	Engineering Change Proposal
FCA	Functional Configuration Audit
FQR	Formal Qualification Review
FSD	Full-Scale Development
GFI	Government Furnished Information
GFM	Government Furnished Material
HOL	High Order Language
HQ AFSC	Headquarters Air Force Systems Command
HQ USAF	Headquarters United States Air Force
IV&V	Independent Verification & Validation
MOL	Machine Oriented Language
OSD	Office of Secretary of Defense
PCA	Physical Configuration Audit
PDR	Preliminary Design Review
PM	Program Memoranda
PMD	Program Management Directive
PMP	Program Management Plan

PO	Program Office
POM	Program Objective Memoranda
RFI	Request for Information
RFP	Request for Proposal
ROC	Required Operational Capability
SDR	System Design Review
SOW	Statement of Work
SPO	System Program Office
SRR	System Requirements Review
WBS	Work Breakdown Structure