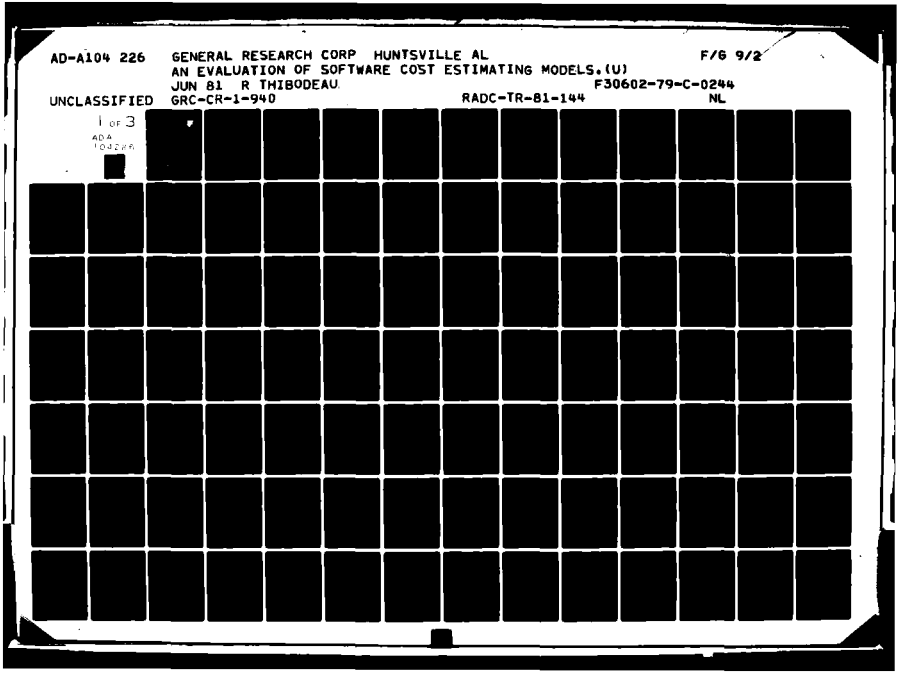

AD-A104 226
GENERAL RESEARCH CORP HUNTSVILLE AL
F/G 9/2
AN EVALUATION OF SOFTWARE COST ESTIMATING MODELS.(U)
JUN 81 R THIBODEAU
F30602-79-C-0244
UNCLASSIFIED
GRC-CR-1-940
RADC-TR-81-144
NL
1 OF 3
ADA
104226

RADC-TR-81-144
Final Technical Report
June 1981

AN EVALUATION OF SOFTWARE COST ESTIMATING MODELS

General Research Corporation

Robert Thibodeau

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

AN EVALUATION OF SOFTWARE COST ESTIMATING MODELS

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

RADC-TR-81-144 has been reviewed and is approved for publication.

APPROVED: Rocco F. Iuorno

ROCCO F. IUORNO
Project Engineer

APPROVED: Alan R. Barnum

ALAN R. BARNUM
Assistant 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 (ISIE) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-81-144	2. GOVT ACCESSION NO. AD-A104	3. RECIPIENT'S CATALOG NUMBER 226
4. TITLE (and Subtitle) AN EVALUATION OF SOFTWARE COST ESTIMATING MODELS.		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report. Sep 79 — Oct 79
		6. PERFORMING ORG. REPORT NUMBER GRC-CR-1-940
7. AUTHOR(s) Robert Thibodeau		8. CONTRACT OR GRANT NUMBER(s) F30602-79-C-0244
9. PERFORMING ORGANIZATION NAME AND ADDRESS General Research Corporation 307 Wynn Drive Huntsville AL 35805		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 55812014
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIE) Griffiss AFB NY 13441		12. REPORT DATE June 1981 13. NUMBER OF PAGES 268
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: Rocco F. Iuorno (ISIE)

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

Software Cost Models	Software Cost Factors
Software Cost Estimation	Software Costing Techniques
Software Life Cycle Cost	Software Economics
Software Productivity Measurement	Software Cost Analysis

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

Nine software cost estimating models are evaluated to determine if they satisfy Air Force needs. The evaluation considers both the qualitative and quantitative aspects of the models' outputs.

Air Force needs for cost estimates are established by the Major Weapon System Acquisition Process. Associated with the different development phases are five cost estimating situations. Decisions that are made

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

early in the Acquisition Process require software cost information that includes the entire life cycle for complete software systems, subsequent decisions require more detailed cost information.

Comparison of the outputs of the nine test models with the requirements established by the five cost estimating situations indicates that the models are able to satisfy only the needs of the earliest phase of the Acquisition Process. The models perform satisfactorily for the purpose of allocating funds for software acquisition, but they fail to support such needs as assessment of alternative designs, proposal evaluation, or project management.

Estimating accuracy was measured by comparing outputs with actual experience using data from three organizations representing 45 software development projects. The best model performance (Relative root mean square error ≐ 40 percent) is obtained when a model is calibrated using representative historical data. Calibration was found to have greater effect on estimating accuracy than the model form.

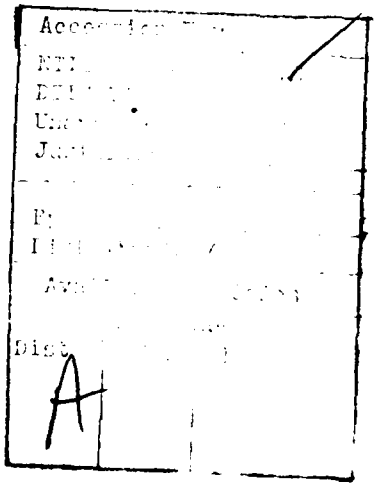


TABLE OF CONTENTS

TABLE OF CONTENTS (Cont)

TABLES

TABLES (Cont)

FIGURES

FIGURES (Con't.)

1 INTRODUCTION AND SUMMARY

1.1 COST ESTIMATING AND SOFTWARE COST MODELS

Cost estimating is an integral part of the Air Force major weapon system acquisition process [1] [2] [3]. The Air Force manages the weapon system life cycle by continually balancing performance, cost, and risk for the system and its components. Throughout the weapon system life cycle it is necessary to estimate the cost of part or all of the system over a part or all of its development and operational life.

Computers are an increasingly important part of Air Force weapon systems in terms of both function and cost [4] [5]. Until recently, most of the cost analysis and planning related to computer subsystems was directed to the hardware. However, increased capabilities and reductions in the cost of hardware have had the effect of increasing the amount of software needed for each system and its cost relative to the cost of the hardware. It is now often necessary to budget large portions of the system life cycle cost to the development and maintenance of these software components [6] [7] [8]. Therefore, more attention is being given to the methods used for making estimates of the resources to be invested in the software subsystems.

A software cost model is a systematic procedure that relates cost to certain variables or cost factors. A number of such models are available to cost analysts. The Air Force has commissioned this study to examine some of these models to learn the extent to which they satisfy Air Force needs and to learn how the quality of software estimating can be improved.

1.2 THE AIR FORCE PERSPECTIVE AND SOFTWARE COST MODEL RELIABILITY

There are cost estimating situations in which the Air Force must consider the effect on software cost of who builds it or how it is built. Therefore, it is useful to divide cost factors into those that describe the product under development and those that describe the manner in which it is built. Cost factors other than those that describe the product are affected

by the selection of a development organization or the development process. These non-product cost factors are difficult to identify and measure. In the case of hardware porducts they include such things as experience, tools, and facilities. Given the proper adaptation of definitions, the same terms are applicable to software development. In either case, these environmental factors may appear explicitly in cost estimating procedures or, more often, they may influence the applicability of a given model to a given development environment in some unknown way. A major consideration in evaluating models for Air Force use is measuring the ability of the model to define the environmental parameters. This is because the Air Force must always make its estimates at arms length. It must know how the cost of software is influenced by how it is developed and who develops it.

It may be helpful to compare methods for estimating software cost with those used for estimating computer hardware cost. Computer hardware cost estimating is more advanced than software cost estimating. This is because there has been a recognized need for it for a longer time and because cost estimating techniques that were developed for other electronic components were adaptable to computers. Hardware possesses readily identifiable measures of size and performance that have been correlated with cost [9] [10]. Given a hardware product with specified physical and functional characteristics, methods exist [11] [12] [13] [14] for considering the effects on cost of non-product factors such as state of the art advance, experience, learning and manufacturing techniques. Therefore, it is possible to make early cost estimates using average industry performance (or some desired increase over the existing average); and then, in later phases of the life cycle, it is possible to evaluate proposals and give proper credit for new approaches and to identify high risk or infeasible concepts.

Although software costs are also affected by non-product factors [15] [16] [17], there are no reliable procedures for quantitatively describing their effects on cost. The most common existing procedure for accounting for differences in development methods or organizational

experience is to base model estimates on historical experience similar to the proposed development environment. However, there is very little objective basis for distinguishing among projects to determine whether they are truly applicable to the proposed environment. This capability is essential if the Air Force is to properly evaluate software development and maintenance proposals from different organizations.

There are several reasons why software cost estimates are not as reliable as those for hardware [18] [19]:

- Software development engineering is a relatively new discipline.
- Software design and development methods have been affected by the explosive development of computer hardware which has changed the cost incentives relating software and hardware.
- Software has only recently become a major cost item in the weapon system life cycle.
- The relationships between cost and generally accepted cost factors are not established.
- Reliable historical data on software costs are almost nonexistant.

None of these deterrents to reliable software cost estimates represents an insurmountable barrier. One purpose of this project is to evaluate a number of existing cost estimating techniques or models to learn how to overcome past problems.

1.3 OVERVIEW OF THE SOFTWARE MODEL EVALUATION

The evaluation design stems from the belief that any evaluation of the merits of different approaches to a given objective (i.e., obtaining good cost estimates) should be based on the comparison of the approaches with some standard. To permit the evaluation to be only a comparison of how the several existing software models are alike and different is an abdication of the evaluator's prerogative to impose the standard of measurement. To look at all existing models, make a list of their

characteristics and then show how each compares with all the others, makes the assumption that the Air Force needs are represented in the study population. It implies that there are no requirements other than those that prompted the designs of the test subjects. Furthermore, it fails to consider whether the existing models have satisfied even their creators' objectives.

A detailed statement of Air Force estimating needs (Section 2.1), establishes objective standards for cost models that avoids features or qualities of existing models that may be expensive or difficult to achieve, and which are not needed. It is then relatively easy to compare model characteristics and evaluation objectives. Since the evaluation is based on satisfaction of needs, this approach provides a ready basis for establishing priorities for possible research programs.

Past comparative studies of software cost models [20] [21] [22] [23] [24] [25] [26] [27], have provided descriptions of model features and discussed different methods for making estimates. Several studies [28] [29] [30] have been published describing estimating experience with the PRICE S model. But there has been no comprehensive analysis of predictions relative to needs nor a comparative analysis of estimating performance using data from different environments. This evaluation compares estimating performance using three different development data sets. This is an important part of the evaluation design because several reports indicate that environment is a significant factor affecting model estimating accuracy [31] [32] [33]. The use of three data sets is intended to help identify model features that are sensitive to environmental change. Controlling these factors should help uncover other determinants of accuracy.

If the objective of the accuracy evaluation was to determine which of the nine models is the most accurate estimator on a given data set, it would only be necessary to execute the models using the same data and

tabulate the difference between the predicted and measured values of the test variable. Such an evaluation, however, would not tell the Air Force whether the measured accuracy would be obtained for all estimating situations or guide future model development by indicating model attributes that contribute to higher estimating accuracy.

The evaluation of model accuracy should address the following considerations:

- The effect of the software development environment on model performance.
- Attributes of the environment that are associated with the best and worst performance of a model. That is, factors that indicate when it is best to use a given model and when it should not be used.
- The effect on the accuracy measurement of incomplete input sets among the test data.
- The characterization of model structures in a way that will help to identify correlations between structural attributes and estimating performance.

1.4 SUMMARY OF THE REPORT

The material in this report is presented in much the same sequence that the evaluation project was completed. The models to be evaluated were selected and analyzed, the evaluation criteria including Air Force cost estimating needs and accuracy were established, data sets were identified and qualified, and finally the evaluation protocol was executed and the results analyzed. Specifically, the pertinent sections of the report are:

2 Descriptions of the Evaluated Models
3 Definition of the Evaluation Procedure
4 The Establishment of the Evaluation Criteria
5 Execution of the Evaluation Procedure
6 Analysis of the Results of the Evaluation
7 Recommendations for Future Model Development

Section 2 presents the general selection criteria used for the models and includes a one-page summary of each model. The models are described according to the three structural types developed in Section 4.2, their method of making their initial and subsequent estimates, and their outputs.

Section 3 explains the evaluation criteria established for Air Force cost estimating needs and the measurement of prediction accuracy. The cost estimating information needs are established by the Major Weapon System Acquisition Process (Section 3.1). Consistent with this process is the Air Force Software Life Cycle and a comprehensive Work Breakdown Structure (Appendix B). The Weapon System Acquisition Process gives rise to five cost estimating situations that should be supported by cost models. The Software Life Cycle defines the set of activities and events that describe the boundaries of the cost estimates. The Work Breakdown Structure establishes the elements of the product within the life cycle phases that must be identifiable by separate cost values. The evaluation of the extent to which existing models satisfy the five estimating situations is made by comparing the model outputs with the requirements in terms of scope and detail.

Estimating accuracy may be measured using different variables. Section 3.2 discusses several alternative methods and explains why the Average Relative Root Mean Square Error was selected.

A large part of the effort spent on the project was devoted to obtaining accurate descriptions of model inputs and outputs (Section 3.1). Most published model descriptions are vague in their definitions of their variables. It is difficult to know exactly which cost elements are included in the model estimates. One common problem was the variations in the use of the most frequently used input: size of code. Many different definitions were encountered.

Section 4.2 describes the three categories used to designate the model structures:

- Regression
- Heuristic
- Phenomenological

Section 4.3 describes the three organizations that contributed data to the evaluation and some of the processes used to obtain and qualify it.

The nine test models are associated with such a large number of different input and output variables that none of the data sets was rich enough to provide measured values of each. Section 4.4 describes how the missing data items were handled.

The results of the evaluation are presented in Section 5. Section 5.1 describes how well the models satisfy the cost information needs established by the five cost estimating situations, the Software Life Cycle definitions and the Work Breakdown Structure. Section 5.2 contains the results of the accuracy measurements. Estimating performance is related to model and environmental characteristics.

The evaluation indicates that the performance of the models tested is very sensitive to the development environment. Within an environment characterized by similar projects, personnel experience and management techniques, the most accurate models achieved an average estimating error of about 25 percent on the basis of the root mean square error. However, a model that exhibits such performance on one data set may demonstrate an average error approaching 100 percent on another. Even within a single environment one of the best performing models has an error range of $\pm$ 50 percent. These error measurements were made after the models were calibrated on the test data sets. Therefore, the accuracy is greater than would be expected when estimating a new project.

These results indicate that in virtually all estimating situations there are factors that are not properly accounted for by the models tested. These factors are affected by changes occurring between environments and within an environment.

The results of the evaluation are summarized as follows:

A comparison of the outputs of the models under investigation with the Air Force estimating needs indicates that:

- The supporting materials for most of the models do not clearly state the elements included in their estimates and are not precise about their definitions.
- The existing models are better able to satisfy information needs early in the acquisition life cycle.
- None of the models included in this study fully satisfy the Air Force need for information either with regard to scope or detail.
- The models tend to be phase oriented and do not properly describe activities that cross phase boundaries. This precludes obtaining data compatible with both management planning (phase related) and product cost (WBS).
- Although most of the models use the summation of program or module sizes to make their cost estimate, only one model studied provides for keeping track of the cost on a component basis and accounts for the cost of system integration. None of the models provide for all four levels of system definition called for in the Work Breakdown Structure (Ref. Appendix B).

Based on the relative root mean square error measure of performance:

- Recalibration* is the primary factor contributing to the differences in estimating performance among the models tested.
- The contribution of model structure* to estimating accuracy is not significant when the models have been calibrated to the development environment*.

* Definitions of these terms are given in Section 1.5.

- The development environment significantly affects the relative performance of the models tested.
- The effect of development environment on estimating performance precludes the possibility of obtaining generally applicable measures of model performance without applying additional controls.
- Models that do not use size as an input may perform as well as those that do.
- The average RMS Error for all tested models is unacceptably large for Air Force estimating purposes.
- The best performance obtained by any group of the models tested is not adequate for Air Force needs.

Caution must be exercised to avoid extending the interpretation of the results of the accuracy measurements beyond the constraints of this study. Section 6 discusses five considerations affecting the reliability of the measurements.

Section 6.1 explains how the development environment affects estimating performance and the rankings of the models.

Section 6.2 considers the effects on the accuracy measurement of errors in the estimated input values.

Section 6.3 describes the methods used to calibrate the models on the historical data sets and the implication for the evaluation.

Section 6.4 explains the use by some models of parameters and variables that can never be measured.

The recommendations for future model development are divided into two parts. Section 7.1 describes needs for new experiments identified during this project. Section 7.2 makes recommendations for better data definition and collection.

1.5 SOME DEFINITIONS

The discussions in this document include several terms that have specific meanings within the context of the evaluation. They are defined here to clarify the presentation of the results.

Model Structure. A cost estimating model is considered to be the specific representation of the model structure and its associated parameters that is to be executed in a given cost estimating situation. A model structure includes imputs, a calculation process and outputs. It is the formal representation of how the outputs are related to the cost driving variables or inputs. In addition to the inputs, which represent the attributes of a specific project or development effort, there are parameters of constants that complete the quantification of the model. The parameters may be obtained empirically from representative past projects or they may be subjective. They determine and represent the universe of environments for which the model is applicable. In some cases, different parameters are given for different estimating situations (e.g. Doty); in others, the models are presented without restrictions on the applicability of the parameters. Two models (PRICE S and SLIM) identify the parameters and provide means for estimating them for any environment.

Throughout this report the term "model" refers to the combination of the "model structure" and values of the parameters. The "model structure" is the representation of the estimating hypothesis. Our ultimate objective is to relate the attributes of the model structure to accuracy.

Calibration. The process by which values of model parameters are obtained for a given cost estimating situation is called "calibration". The calibration of a model structure may be performed using formal curve fitting methods on a representative historical data set, by using an execution mode of the model, or by selecting values from experience. An important consideration in this evaluation was the proper selection of representative data and methods for calibrating the model structures.

Environment. This is a general term used to describe the source of influencing forces that are external to the product being developed. As was mentioned before, it is conceptually helpful when analyzing model structures to divide the cost-driving factors into two groups: factors that describe the product and are therefore unchanged by how or where the development is completed; and factors that affect the resources needed to develop the product but are independent of its characteristics. The first group are usually referred to as input variables and the second group constitutes the environmental parameters. Examples of environmental factors are: type of development organization, type of contract, method of project organization, development methods, supporting software, facilities, and description and availability of computer hardware.

2 MODEL DESCRIPTIONS

Software cost estimating models were selected for evaluation for one or more of the following reasons:

- Possessing a unique structure
- Representing a common type of structure
- A representative choice of input variables
- A unique choice of input variables
- Widespread use
- Otherwise interesting to the Air Force.

The following models were evaluated:

- Aerospace Corporation
- Boeing Computer Services
- DoD Micro Estimating Procedure
- Doty Associates, Inc.
- Farr and Zagorski
- PRICE S
- SLIM
- Tecolote Research Corporation
- Wolverton

Detailed descriptions of the models including their inputs and outputs are presented in Appendix A. The following are one-page summaries of the models (Table 1) that describe the characteristics upon which inferences concerning the contribution of model structure to performance are based. These attributes include:

- Model type
- Estimating Procedure
  - Level of initial estimate
  - Method of making initial estimate
  - Method of making subsequent estimates
- Characterization of productivity
- Outputs

AEROSPACE CORPORATION

STRUCTURE

Type. Regression

First estimate. Development effort.
Single parameter

Subsequent estimates. No further breakdown of effort.

Development effort is calculated given the number of instructions using an estimating equation of the form:

$$MM = aI^b$$

where MM = Manmonths of development effort

I = Number of instructions

a,b = Constants

OUTPUTS

Effort.

Scope. Assumed to be Analysis through System Test.

Detail. System or CPCI level.

Table 1 Summary of Model Characteristics

BOEING COMPUTER SERVICES

STRUCTURE

Type. Heuristic

First estimate. Development effort.
Multi-parameter

Subsequent estimates. Allocations using fixed ratios followed by phase-related adjustments.

The system is divided into five types of software and the number of delivered instructions is estimated for each component. The system development effort is obtained by multiplying the productivity rate in manmonths per instruction for each type of software and adding the values for the components. The development effort is divided into six life cycle phases using fixed ratios. The phase estimates are adjusted for certain development and software characteristics and recombined to form a revised total development effort.

OUTPUTS

Effort.

Scope. Analysis through System Test

Detail. System level

Table 1 (Cont) Summary of Model Characteristics

DOD MICRO PROCEDURE

STRUCTURE

Type. Heuristic

First estimate. Portion of development effort (Direct development effort)
Multi-parameter

Subsequent estimates. Fixed ratios

Net development effort is calculated using an estimating equation that includes software function and complexity variables along with experience measures.

A constant factor is used to estimate gross development effort which then divided into phases using ratios.

OUTPUTS

Effort.

Scope. Analysis through Installation

Detail. System level

Table 1 (Cont) Summary of Model Characteristics

DOTY

STRUCTURE

Type. Regression

First estimate. Development effort.
Multi-parameter

Effort is related to size and type of code by estimating equations. For small systems the effects of 14 environmental parameters are included using a product function.

OUTPUTS

Effort.

Scope. Detailed Design through Coding and Checkout

Detail. Total effort for a CPCI

Development time.

Table 1 (Cont) Summary of Model Characteristics

FARR AND ZAGORSKI

STRUCTURE

Type. Regression

First estimate. Development effort.
Multi-parameter

Subsequent estimates. No further breakdown of effort

Effort is related to 5 predictor variables by an estimating equation.

OUTPUTS

Effort.

Scope. Detailed design through coding and checkout

Detail. Total effort for a CPCI

Table 1 (Cont) Summary of Model Characteristics

PRICE S

STRUCTURE

Type. Heuristic

First estimate. Portion of development cost (design cost) Multi-parameter

Subsequent estimates. Functional relationships

Cost is related to predictor variables by Tables and equations that are either subjective or empirically derived.

Cost and effort are related by cost per unit time values that are constant for a given phase.

OUTPUTS

Cost.*

Scope. Detailed Design through Installation

Detail. Three phases, Design Implementation Test and Installation. For each phase by activities system analysis, programming, documentation, management, quality assurance. Model options include independent V&V, system integration.

Time.

Computer units.

* Alternative outputs are manhours or manmonths.

Table 1 (Cont) Summary of Model Characteristics

TECOLOTE

STRUCTURE

Type. Regression

First estimate. Development effort.
Single parameter

Subsequent estimates. No further breakdown of effort.

Development effort is calculated using a cost estimating equation with number of instructions as the independent variable.

OUTPUTS

Effort.

Scope. Requirements through Operational Demonstration

Detail. System or CPCI level.

Table 1 (Cont) Summary of Model Characteristics

SLIM

STRUCTURE

Type. Phenomenological

First estimate. Development cost.
Multi-parameter, (linear programming)

Subsequent estimates. Allocations using fixed ratios

Effort is related to predictor values using the "software equation." This along with constraints on time, effort and cost define a range of acceptable solutions (if any).

Cost and effort are related by a constant value of cost per unit.

OUTPUTS

Effort.

Scope. Detailed design through installation for the primary output. Additional outputs include analysis effort.

Detail. System level

Time.

CPU Time.

Documentation.

Table 1 (Cont) Summary of Model Characteristics

WOLVERTON

STRUCTURE

Type. Heuristic

First estimate. Development cost.
Multi-parameter

Subsequent estimates. Allocation using fixed ratios

Cost is related to routine size and category by a constant cost per instruction for each category of software.

OUTPUTS

Cost.

Scope. Analysis through Operational Demonstration

Detail. Seven phases, each with up to 25 activities an eighth phase, Operations and Maintenance has allocations among the 25 activities, but there is no guidance for allocating the eighth phase from the total.

Computer cost.

Table 1 (Cont) Summary of Model Characteristics

3 EVALUATION CRITERIA

The Air Force needs reliable procedures for estimating software costs to support its activities as the manager of weapon system development. It is necessary when examining methods for making cost estimates to be mindful of the Air Force's perspective as the system development manager. The Air Force does not develop system components itself. When estimating the cost of developing and operating a new system, it must at first consider industry-wide capabilities as represented by experience with similar weapon systems. This representation of development performance is adequate for conceptual studies, but it is not valid for evaluating proposals for specific subsystems to be built by specific organizations. For example, a single organization may obtain good results using a given method of cost estimating; but it must be recognized that many variables such as experience, support facilities, and management techniques are relatively fixed in that organization. Their influence on any estimates made by that organization are minimal. However, if the method were adopted by the Air Force and applied to many organizations such as might occur in a major weapon system development, the results may not be satisfactory at all. The model evaluation was designed to look at software cost estimating from the Air Force's point of view.

The choice of evaluation criteria was affected by the following considerations:

- A number of different software cost estimating models already exist.
- Proponents of the models offer testimonials based on their particular experience and estimating needs.
- There is no model or approach that is not without both supporters and critics.
- Much of the existing literature claims there is no reliable method of making software cost estimates.

Given the conflicting evidence it seemed reasonble to conduct an evaluation of representative cost models to address the following:

- The needs of the Air Force for software cost estimates.
- The extent to which existing software cost models satisfy those needs.
- The characteristics of existing model structures that make them good or bad performers for Air Force purposes.
- Methods for improving the quality of future Air Force software cost estimates.

The evaluation was divided into two parts:

- The satisfaction of Air Force needs for software cost estimates in terms of specific items of information.
- The realization of estimates with accuracy acceptable for making decisions concerning selections of alternative design concepts, allocation of resources, and managing the software life cycle.

This section of the report describes how criteria were defined that establish the Air Force needs for cost model performance in terms of items of information and accuracy.

The first subsection describes the Major Weapons System Acquisition Process, the Software Life Cycle and the Work Breakdown Structure. It then shows how these lead to five cost estimating situations which are described in terms of scope of the life cycle addressed, level of detail in the estimates and the desired estimating accuracy.

The second subsection establishes the criteria for measuring estimating accuracy; and the final subsection discusses some evaluation criteria that were considered but not included.

3.1 INFORMATION NEEDS

The Air Force identifies two types of computer system development. One is the creation of computer systems that are end products. That is,

they perform a separate function. These are for the most part management information systems. The other type of computer system is an integral part of a larger system. It is characterized by stringent and complex interfaces with its environment. These are usually called, "embedded systems."

For the purpose of this evaluation, the needs for software cost information will be established by the process governing the development of embedded software. However, this should not limit the applicability of the results. For one thing, most of the models are used for both types of development; and, for another, the software portion of the development cycle is nearly the same for both types of systems. The embedded system development must be governed in addition to its own requirements by the needs of the weapon system.

The representation of both the software life cycle and its controlling environment, the weapon system life cycle, allows us to specify the needs for software cost estimates considering the points of view of the weapon system and the software components. The weapon system manager must know how the needs of the software components will affect the cost, schedule and risk of the weapon system. He must also know how the performance of the weapon system in terms of functions, speed, reliability, etc. are affected by the software system cost, schedule, and risk. When the software resources have been allocated, the software subsystem manager must assess his cost, schedule, and risk in terms of lower level design choices. He as well as the weapon system manager must make preliminary cost-performance trade-offs, prepare statements of work, evaluate proposals, and monitor contracts.

The following sections describe the evolution of the weapon system definition as it occurs during the Major Weapon System Acquisition Process. The aspects of the weapon system that establish the software requirements are highlighted. The software life cycle is presented along with the definition of the characteristics that contribute to the estimation of its cost, schedule, and risk.

The Acquisition Life Cycle for Major Defense Systems is the formal decision process regulating the acquisition of electronic systems that include software. Electronic Systems are one of seven types of system identified in MIL-STD-881A, Work Breakdown Structures for Defense Material Items [34]. The acquisition of computers and software that are embedded in a weapon or command and control system are normally governed by the Air Force 800 series of regulations.

AFR 800-2 defines the Acquisition Life Cycle for Major Defense Systems as normally comprising five sequential phases (Figure 1):

- Conceptual
- Validation
- Full-Scale Development
- Production
- Deployment

Review by the Defense Systems Acquisition Review Council (DSARC) normally follows each of the first three phases and Secretary of Defense approval is required to proceed from one phase to another. There is some flexibility in the composition of the phases. In general the process is designed to insure:

- Continuing operational need
- Adequate system performance
- Acceptable cost
- Favorable cost effectiveness relative to other alternatives

A Decision Coordinating Paper (DCP) is prepared to support each DSARC review.

The procedure used as the basis for the definition of need is taken from [34] which is based mainly on interpretations of:

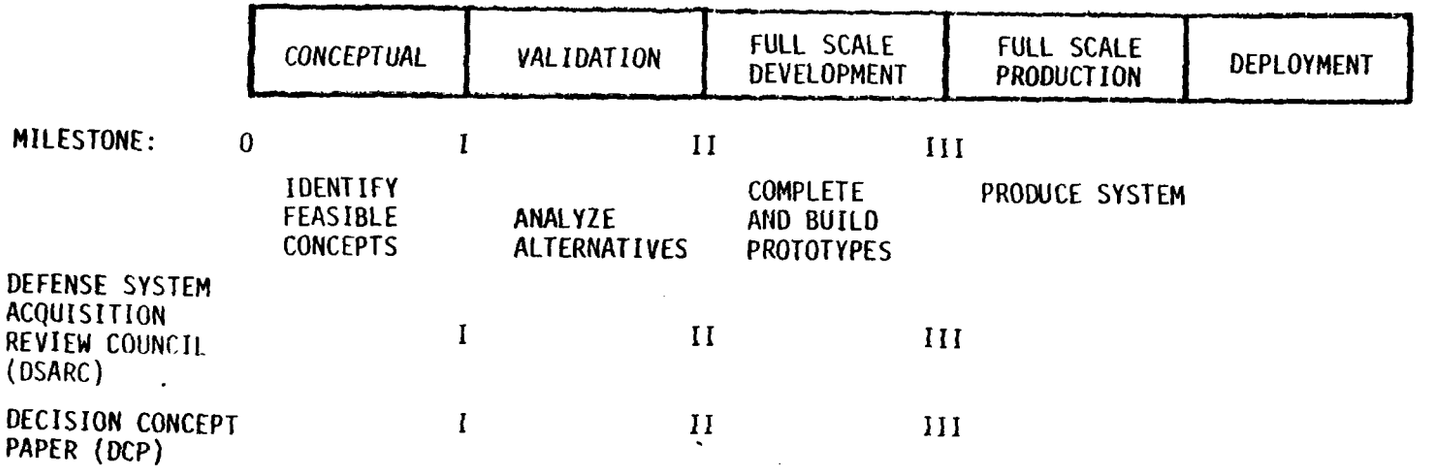


Figure 1 Major Weapon System Life Cycle

AFSCP 800-3
AFR 800-14, Vol. II
AFR 800-2
DoDI 5000.2

Summary of the Development Phases.

Conceptual Phase.

1. Explore, formulate, and evaluate possible system requirements.
2. If necessary, devise an optimum, affordable, and cost effective preferred approach to the system's development, production, and deployment.

Considerable preliminary design and analysis of software may be necessary to support these objectives. Demonstration, prototype and simulation software may be required. Conceptual Phase design and analysis should be limited to whatever is necessary to establish technical feasibility and credible estimates of costs and development times. Design and analysis should be most detailed where technical risk is greatest.

The Conceptual Phase has no prescribed time limit. Before DSARC review of the draft DCP begins, the program can be terminated with the approval of the highest command level which authorized it. Once DSARC review begins, the Conceptual Phase will normally end with the Secretary of Defense's Program Decision to proceed into the Validation Phase (with or without specific redirection), or to end the program.

Validation Phase.

1. Assess the preferred design approach selected during the Conceptual Phase by comparing it with the Initial System Specification.
2. Rectify any difficiencies or develop a new approach if necessary.
3. If and when a sound system design approach is achieved, provide sound technical, contractual, economic, and organizational bases for the Full-Scale Development.

Most Validation Phase work is to demonstrate the feasibility of doubtful components and subsystems and interface definitions, and to improve estimates of performance cost and schedule. All can be considered risk-reduction measures.

The Validation Phase may also include contracted design competitions.

The Validation Phase is intended to reduce risk significantly and to allow negotiation of clear contracts for the subsequent acquisition phases. The development of unambiguous specifications and testable requirements is most important.

Full-Scale Development.

1. A working prototype of the system (or the system if there are no replicas).
2. Test results proving that this prototype can meet its functional and performance requirements.
3. A Cadre trained in the system's operation and maintenance.
4. The documentation needed to begin the system's Production Phase (if any) or otherwise needed for its Deployment Phase.

For the system's software the Full-Scale Development Phase is intended to yield the initial operational versions of the computer programs, not prototypes.

The system's operational software (i.e. the executives and applications programs necessary to meet the system's operational requirements), plus the support software necessary to build and maintain the operational software and to support the Design, Test, and Evaluation and Initial Operational Test and Evaluation functions must normally be completed during Full-Scale Development.

If proprietary software is to be incorporated into the system, the Government must decide whether the price represents an advantage over contracted development.

Production Phase.

Activities are limited to maintenance and modification of existing software. They may also include site-specific testing and installation.

Software has a life cycle of its own (Figure 2) that exists in concert with the weapon system life cycle. Software requirements for embedded subsystems are established primarily by the needs of the Weapon System.

Table 2 [34] describes the activities and products comprising the Software Life Cycle.

The functions assigned to the software comprise, along with the definition of the computer elements, the basis for estimating the time, effort, and other resources required to create the software and test it. If the investment needed to provide the prescribed software functions are not acceptable, then a redefinition of the allocation of functions among hardware and software may be necessary. If this doesn't resolve the conflict, it may be necessary to revise the requirements.

This iteration between software requirements and feasibility is continuous throughout the development phases. Problems thought solvable during the Concept Phase may later prove not to be. Sometimes the software definition and design process must go on for some time before negative re are obtained.

development of systems that contain software is an iterative proc the steps of the software life cycle are an integral part of the system life cycle. Figure 3 [35] describes the combined system-software life cycles.

Figure 2 The Software Life Cycle

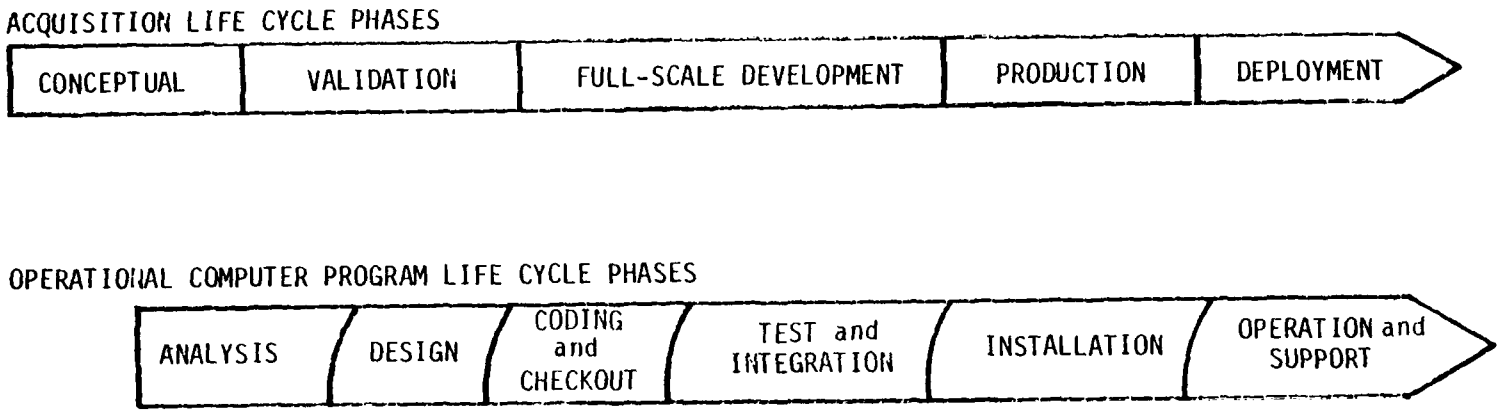


Figure 3 Relationship Between Weapon System and Software Life Cycles

TABLE 2 SOFTWARE LIFE CYCLE ACTIVITIES AND PRODUCTS

ANALYSIS PHASE

Activity	Product(s)
A. Devise & analyze alternatives for the system, Segment (if any), or any Software Subsystem directly containing the Computer Program.	A.1. Tradeoff study reports 2. Initial or Authenticated System Specification & Segment Specification (if any).
B. Allocate requirements to the Computer Program: i.e., Functions. Performance (e.g., response times). Interface (with others). Design Constraints (e.g., prescribed algorithms, core & processing time budgets). Testing.	B.1. Authenticated Development Specification for each CPCI. 2. Possible higher-level specification, and ICD, changes. 3. Parts of draft Product Specifications containing design approaches for each CPCI.
C. Conduct PDR(s) for the Computer Program's CPCI(s).	C. PDR minutes and action item responses.

DESIGN PHASE

Activity	Product(s)
A.1. Define algorithms not previously prescribed. 2. Design data storage structures. 3. Define Computer Program logic.	A.1. Functional flowcharts. 2. Detailed flowcharts. 3. Data Format descriptions. 4. Description of algorithms not previously prescribed.
B. Allocate Computer Program requirements internally (e.g., to CPCs)	B. Preliminary Product Specifications, including the above.
C. Test Planning.	C.1. System, Segment (if any) and CPCI Test Plans. 2. Preliminary CPCI Test Procedures.
D. CDR(s) for the Computer Program's CPCI(s).	D. CDR minutes & action item responses.

TABLE 2 (Cont) SOFTWARE LIFE CYCLE ACTIVITIES AND PRODUCTS

CODING AND CHECKOUT PHASE

Activity	Product(s)
A. Coding.	A-B. Code.
B. Limited checkout of compiler or assembly units.	
C. Corresponding logic & data structure revisions.	C. Altered Product Specifications, including compiler/assembly listings.

TEST AND INTEGRATION PHASE

Activity	Product(s)
A. Test Planning.	A.1. Final CPCI Test Procedures. 2. Segment (if any) and system-level Test Procedures.
B. Module tests.	B-D.1. Test Reports. 2. Computer Program coding changes. 3. Modified Product Specifications. 4. Possible high-level specification, and ICD, changes.
C. CPCI tests (PQT & FQT).	
D. Software Subsystem integration.	

INSTALLATION PHASE

Activity	Product(s)
A.1. DT&E of any Segments. 2. System-level DT&E.	A.1. Segment (if any) Test Reports. 2. System-level DT&E Test Reports. 3. Computer Program coding changes. 4. Modified Product Specificaitons. 5. Possible higher-level specification, and ICD changes.
B. Site Adaptation (if any).	B.1. Possible site-specific coding changes. If so: 2. Version Description Documents & 3. Test Reports.
C. IOT&E	C. IOT&E Test Reports.

TABLE 2 (Conc) SOFTWARE LIFE CYCLE ACTIVITIES AND PRODUCTS

OPERATION AND SUPPORT PHASE

Activity	Product(s)
A. FOT&E	A. Analogs of Test and Integration Phase products.
B. Construction, installation, & checkout of software maintenance & training facilities.	B. Related documentation.
C. Software maintenance & modification.	C.1. New software Versions 2. Version Description Documents. 3. Possible specificaiton changes. 4. New or revised Test Plans and Test Procedures. 5. Additional tests. 6. Additional Test Reports.

Abbreviations

CDR	Critical Design Review
CPC	Computer Program Component
CPCI	Computer Program Configuration Stem
DT&E	Development Test and Evaluation
FOT&E	Follow-On Operational Test and Evaluation
FQT	Formal Qualification Test
ICD	Interface Control Drawing
IOT&E	Initial Operational Test and Evaluation
PDR	Preliminary Design Review
PQT	Preliminary Qualification Test

Source: [34]

Software cost models must contribute to a rapid determination of economic feasibility of the software components. Ideally a model will help integrate time, effort and risk in order to establish feasibility. It will do this using information describing the system attributes.

In the following discussion, the weapon system acquisition cycle is used to define the cost estimating needs and the software life cycle is used to describe the elements of software system cost.

Table 3 [34] details the weapon system and related software subsystem activities comprising the acquisition life cycle.

Analysis of Table 3 indicates a continuous transition in the needs for estimates over the development cycle. During the early phases the need is for high level or aggregated estimates of development time and cost for any number of alternative design concepts. As the system design matures, its elements become defined at lower levels and each has a greater number of attributes. The individual element has a more limited role in the system, but it is described in greater detail and it must function in concert with many other elements. Initially, we might speak of the navigation subsystem and its functions. Later, we would describe the alignment element with its functions, speed, accuracy and interfaces with the accelerometers, gyros, etc. Therefore, inherent with the process of increased system definition is the need to describe levels of intergration and interface in addition to component attributes.

The need for software cost and resource estimating during the development life cycle proceeds from the rapid calculation of gross estimates for several concepts to rather detailed estimates devoted to a single design. Ideally the estimating methods needed to support this process would be functionally oriented in the early phases and evolve to variables describing design characteristics in the end phases.

TABLE 3

ESTIMATING NEEDS FOR SOFTWARE LIFE CYCLE PHASES

PHASE	ESTIMATE	LEVEL OF DETAIL	IN SUPPORT OF
Conceptual	Development time	Total Development	Evaluation of statement of need.
	Cost		
	Risks		
			Preliminary system design studies (e.g., analysis of design alternatives)
	Computer time for testing	Total Testing Time	Preliminary Test Estimates
	Testing Manpower	Effort by type of personnel	
	Effort, times and cost to produce and test development software	Total Development	Preliminary Cost and Schedule Estimates
	Development Schedule		
	Costs, Schedules, Risks	WBS* level 1	Formal cost effectiveness analysis Optimizing alternatives in terms of performance, schedule and WB/C limits
	Risk evaluation or Cost/Schedule Studies	WBS 1	Initial draft DCP I
	Cost and Schedule	WBS 1	Initial System Specification

* Work Breakdown Structure. See Appendix B for definitions of levels.

TABLE 3 (Con't)

PHASE	ESTIMATE	LEVEL OF DETAIL	IN SUPPORT OF
Validation	Development Time Cost	WBS Level 2	Revised system specification
	Manpower	WBS Level 2	Revised PMP
	Development time Effort Cost	WBS Level 2	RFP including Model Contract
	Risk Assessments	WBS Level 2	Allocated Baseline and higher-level Specifications
	CPDP Manpower and Schedules Training Manpower and Schedules	WBS Level 2	Initial CRISP
Full-Scale Development	Cost and Schedule	WBS Level 3	RFP Release and Proposal Analyses Revised Allocated Baseline
	Resource Allocations, Impacts, Changes	WBS Level 3	ECP requests and analyses

The need for precision goes through a similar evolution. During the early phases it is only necessary to determine if a concept is totally out of reach in terms of cost or development time. Subsequently it is necessary to weigh the cost and risk of one design concept relative to another. The final estimates involve the commitment of funds and personnel and demand the greatest possible precision.

Table 3 has been used to prepare descriptions of five cost estimating situations that represent the different kinds of estimates described above. These descriptions (Table 4) which include the scope and detail of the estimates and their accuracy are the basis for evaluating how well each model satisfies the needs for cost information during the weapon system life cycle.

Having described the general need for cost information in terms of life cycle scope and detail, it is possible to extend the criteria in Table 4 to include the Work Breakdown Structure (Appendix B). This extension provides a means of precisely describing all the software estimating needs. Each major element of the WBS (Level 1) is divided into appropriate measures of the software product (Level 2). The relationship between the two levels is shown in Table 5.

Appendix B describes three WBS levels. Table 3 indicates that program management needs extended to the third level. However, none of the models evaluated provide cost estimates in such detail. Carrying this detail in the evaluation process is a needless complication considering that none of the models can provide the information. Therefore, the third level of the WBS is considered to be a description of the data that should be included in the higher level estimates. The first two levels will be the only ones considered in the remainder of the report.

Figure 4 depicts the software cost elements in graphical form. The columns represent the Software Life Cycle phases and the rows represent

TABLE 4 FIVE COST ESTIMATING SITUATIONS

1. Conceptual Phase, Cost Feasibility

NEED

In support of the analysis of perceived deficiencies in existing systems, estimate software component costs and development times for defined alternatives.

SCOPE

Total life cycle cost, Conceptual through O&S.

LEVEL OF DETAIL

Weapon System - Total cost of all software components.

INPUTS

System performance and functions.

LEVEL OF PRECISION

± 30%

EXAMPLE

The software-related costs for a new interceptor aircraft.

TABLE 4 (Cont) FIVE COST ESTIMATING SITUATIONS

2. Conceptual Phase, Preliminary System Design Studies

NEED

Support the evaluation of functional allocations for system components by estimating software development time and cost.

SCOPE

Cost of defining, designing, producing and owning major software components.

LEVEL OF DETAIL

System functional components.

INPUTS

System segment performance, preliminary performance allocations, preliminary size and system interface descriptions.

LEVEL OF PRECISION

± 25%

EXAMPLE

Compare the software development time and cost for a four function versus a five function navigation system.

TABLE 4 (Cont) FIVE COST ESTIMATING SITUATIONS

3. Conceptual Phase, Preliminary Contract Cost and Schedule Estimates

NEED

This need is repeated as necessary to support the evaluations of alternatives leading to DCP I. The level of detail remains fairly constant although some analyses may require defining critical components to more detail than the others. The only thing really changing is the confidence in the results. System components are defined by function and performance.

SCOPE

Validation through O&S.

LEVEL OF DETAIL

First level WBS for each software component

INPUTS

Software functions, performance, interfaces, inputs, outputs.

LEVEL OF PRECISION

± 20%

EXAMPLE

Estimate the development time and cost for a real time display system to be let out for bids.

TABLE 4 (Cont) FIVE COST ESTIMATING SITUATIONS

4. Validation Phase, Support of Validation Phase Contracting

NEED

Allocate funds, support RFP preparations and assist in software related proposal evaluations for Validation Phase contracts.

SCOPE

Software system design through O&S.

LEVEL OF DETAIL

Software WBS level 2, system segment.

INPUTS

CPCI characteristics and performance.

LEVEL OF PRECISION

± 15%

EXAMPLE

Estimated cost including facilities, training, etc for the weapon delivery software for a fighter-bomber.

TABLE 4 (Conc) FIVE COST ESTIMATING SITUATIONS

5. Full Scale Development, Evaulate Progress

NEED

Monitor the progress of software system components during development.

SCOPE

CPCI design through O&S

LEVEL OF DETAIL

Software WBS level 3, CPCI and CPCR.

INPUTS

CPCI and program functions and performance.

LEVEL OF PRECISION

± 10%

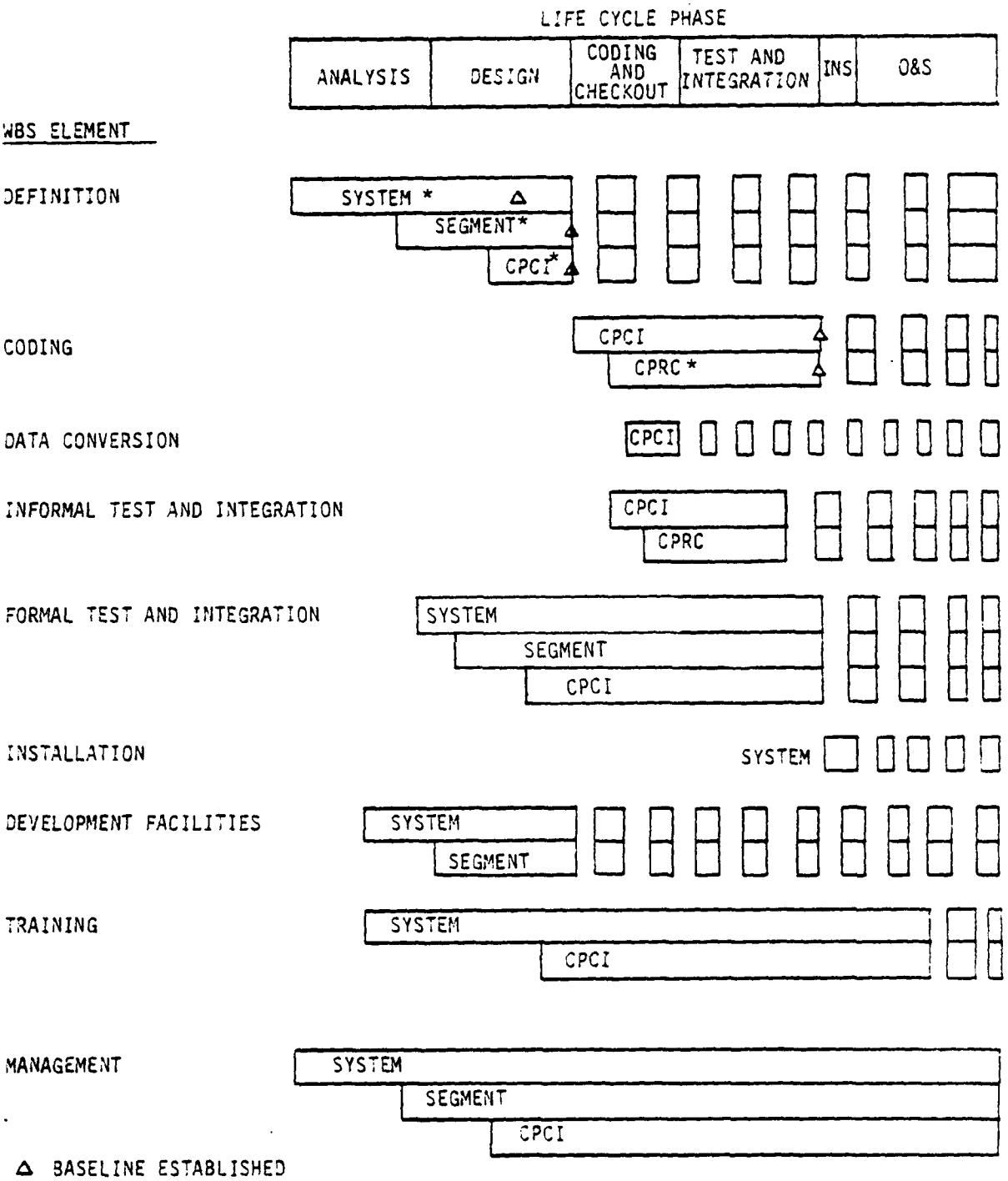
EXAMPLE

Prepare management decision boundaries for cost and schedule for a software development project under contract.

TABLE 5

DECOMPOSITION OF SYSTEM ELEMENTS BY MAJOR WORK BREAKDOWN STRUCTURE DEFINITIONS

	LEVEL 2			
	SYSTEM	SEGMENT	CPCI	CPRC
LEVEL 1				
DEFINITION	X	X	X	
CODING			X	X
DATA CONVERSION			X	
INFORMAL TEST & INTEGRATION			X	X
FORMAL TEST AND INTEGRATION	X	X	X	
INSTALLATION	X			
DEVELOPMENT FACILITIES	X	X		
TRAINING	X		X	
MANAGEMENT	X	X	X	



△ BASELINE ESTABLISHED

* Definitions of these terms are given in Section 3.1.

Figure 4 The Definition of the System Elements and Their Relationship to the Software Life Cycle and WBS

the elements of the Work Breakdown Structure (WBS). The WBS elements are identified according to the physical decomposition of the software system.

The software system is composed of the following elements:

- System A body of software that performs an identified function in the weapon system. It is complete and distinguishable from other bodies of software.
- Segment A major subsystem or component of a system usually identified with a specific function.
- Computer Program Contract Item (CPCI) A body of software identified for acquisition by separate contract. In large systems it is usually part of a Segment. In smaller systems a CPCI may be equivalent to a Segment or even a System.
- Computer Program Reporting Component (CPRC) In large systems this represents a body of software defined for purposes of configuration control and program management.

Figure 4 indicates that the system cost elements may cross life cycle phase boundaries. It is important to depict this relationship because many cost models do not make clear distinctions between the WBS elements and phase costs.

Figure 4 represents a detailed template for depicting the estimating needs represented by the five cost estimating situations in Table 4. If the cost estimating situation calls for a system level estimate that includes the entire life cycle, the ideal cost model would be shaded at the system level for the entire row. WBS elements stated at the lower levels such as Coding and Data Conversion would be cross-hatched to indicate that they are included in the estimates at the system level of aggregation.

In Section 5, Results, Figure 4 is used to describe each model's outputs. In that section a summary is presented that indicates how well

the model satisfies the needs associated with the five cost estimating situations.

3.2 ACCURACY

Total effort was selected as the performance measure for evaluating model prediction accuracy. The selection was made because of it is relatively easy to justify and interpret. It was also done to fix attention to the single model output that everyone should agree is the most important indicator of model prediction accuracy.

It is possible to envision several alternatives for specifying the accuracy of a model estimate. For example, we might use estimated values of the costs of the life cycle phases to construct a weighted estimating function. The function values obtained from the outputs of the models for a given project would be compared with the value obtained from actual measurements to produce error measures. The weights in such an approach might also be obtained from the test data sets. Error functions could also be constructed from the different types of output information such as effort, computer time and facilities.

Total effort was chosen rather than cost because most of the models being evaluated calculate effort and because the available historical data are in terms of effort. However, the use of effort is desirable also because it avoids the need for adjusting estimates for variations in the value of the monetary unit and the problem of measuring overhead and indirect costs. These items vary significantly from one organization to another.

Unfortunately, it is not possible to specify a uniform basis for the total effort measurement. As can be seen in the Results section, the different models do not include the same scope of life cycle activities in their estimates. Therefore, the measurement of prediction accuracy had to be applied to the primary span of the model prediction. In most cases this means the prediction that is constructed using the primary elements of the model structure. Some models use these primary estimates to

compute other phases from fixed ratios. We believe that the performance of the model structure is better represented by the initial estimate in such cases. Having selected the basis for measuring model estimating performance, it remains to define the way to use the measurement to obtain comparisons among the models.

Mean Proportional Error. The ratio of actual to estimated project size describes the error as it relates to the estimate. It is directly transformable into the percentage error of the estimate itself. Being a proportion it allows larger errors in larger projects, but this is acceptable because we tend to think in terms of percent error rather than absolute difference. A 10 manmonth error in a 1000 manmonth project is not as important as the same error in a 6 manmonth project. The disadvantage of the MPE as it is formulated is that it becomes compressed by estimates that are large compared with the actual value. This makes the standard deviation small when taken over a given data set. To reverse the numerator and denominator results in a similar weighting when comparing samples containing large projects with samples made up of small ones.

Average Error. The average difference between the estimated and actual effort taken over a data set presents a measure of accuracy that is not weighed by either the size of the estimate or of the actual measure. This avoids the problem associated with the mean proportional error, and dividing the average error by the mean project size in the test sample provides scaling for the measure.

Root Mean Square Error. A characteristic of some software cost models is their tendency to produce estimates that are very different from the actual experience under certain conditions.

It was decided to select an error measurement scheme that penalizes such extreme behavior. The root sum square error measure provides such a penalty and therefore it was selected as the method for making comparisons among the models. The RSS error is divided by the average project size in the sample set for scaling. The measure is defined to be:

$$\frac{RMSE}{\overline{ACT}} = \frac{\left[\frac{1}{N}\sum_{i=1}^{N} (ACT_i - EST_i)^2\right]^{1/2}}{\frac{1}{N}\sum_{i=1}^{N} ACT_i}$$

Where:

ACT_i = The measured size of the ith project in the sample set.

EST_i = The estimated size of the ith project.

N = The number of projects in the sample.

3.3 OTHER EVALUATION CRITERIA

In addition to the information provided and the prediction accuracy of software cost estimating models, there are a number of model attributes that would influence the decision to select one model structure over the others. These would include:

- Data needed to execute the model.
- Effort needed to execute the model.
- Time required to obtain estimates.
- Total cost of estimates.

Information is presented that would allow anyone to make inferences regarding a model's ranking regarding such criteria. However, no attempt was made to compare the models according to them for the following reasons:

- The criteria are difficult to measure. Any weighting of the attributes to obtain a composite score would be arbitrary.
- Any deficiencies would be model specific and the evaluation is concerned with the performance of model structures.
- A model that provides the information needed and does it accurately would be preferred no matter how badly it scored on the other measures.

The findings in this report should indicate how well one model structure performs in the test environments compared with the others evaluated. If two models satisfy the primary criteria equally well, it is a simple matter to observe the other attributes that may be important to an individual organization such as ease of execution, data, cost, etc. These considerations would have different importance to using organizations.

4 EVALUATION PROCEDURE

The following steps were executed in performing the evaluation of the cost estimating models:

1. Select models for inclusion in the evaluation.
2. Obtain model descriptive materials.
3. Analyze definitions of model input and output variables.
4. Prepare model descriptions.
5. Classify models by type.
6. Prepare list of input and output variables.
7. Compare model outputs with established evaluation criteria for needed information.
8. Construct test data sets.
9. Analyze definitions of items in test data sets.
10. Establish means for estimating missing input data items.
11. Prepare input data.
12. Execute models.
13. Calculate comparative estimating accuracy according to established accuracy evaluation criteria.

Several steps presented problems or require some explanation:

- Definitions of model and data set variables.
- Model types.
- Test data sets.
- Missing input data items.

4.1 DEFINITIONS OF MODEL AND DATA SET VARIABLES.

A problem that continues to limit the development of accurate software cost models is the lack of good quality data with which to test theories describing the relationships between cost and predictor variables. An important aspect of data quality is the enunciation and consis-

tent application of definitions of the variables describing the software, its development environment, and its performance. A substantial amount of effort was spent during the evaluation of the models to minimize the adverse effects arising from discrepancies in data definitions. The following paragraphs will examine in detail two important model variables: software size and development effort.

Size Definitions. Eight of the nine models included in this evaluation use size as an input, yet problems occur when trying to determine precisely what a value of the size attribute represents. Often researchers either through oversight, lack of precise data or ignorance of the problem fail to specify the size measure completely. For example, number of source statements and number of object instructions are two of the terms that are frequently presented to software developers in questionnaires without further qualification. As a result it is possible to obtain historical data that are internally inconsistent by 100 percent because of the vague definitions. Obviously if a model is being used for predictions and the inputs are off by such an amount the estimates will be similarly affected. It is likely that in many cases neither the person supplying the historical data nor the cost analyst realize there are differences in interpretation of the data.

Consider the deliberation of a programmer who is being asked the size of one of his programs by questionnaire. The question is: "Number of Source Statements." If the programs are written say in FORTRAN, the compiler will normally give a count of the number of lines in the program which in most cases will be equal to the number of statements. Most FORTRAN compilers limit one statement per line. If the statement is spread over several lines for clarity, the compiler still counts it as one line. For the most part the FORTRAN programmer has a ready source for his response to the question.

At the other end of the scale is the programmer writing in a freely structured language such as COBOL. His response is considerably more difficult. Such language structures use punctuation to delimit statements and therefore a line of code may have several statements or a statement may stretch over several lines. Since COBOL compilers do not usually indicate the number of statements in a program, but only the number of lines, the programmer has no easy source for the requested information.

The conscientious programmer may make this problem known to the cost analyst who may or may not be in a position to address it. More often the programmer will assume the question calls for lines of code or he may make some arbitrary judgement about the relationship between statements and lines. In either case there is considerable opportunity for error in the capture of the most commonly used predictor of program development cost.

There are other problems in interpreting the term "source code statements" as the descriptor of program size.

Most higher order languages permit the inclusion of comment statements throughout the source code. These statements usually describe what the program is doing at various points. Some programmers write many comments. Large programs exist in which there are twice as many comment lines as code lines. Other programmers do not write any comment lines. Even within a single group a large variation exists among programmers. A programmer who normally comments a program extensively may do so very sparingly if he is being pressed to complete the program on a tight schedule.

Some cost models treat comments the same as lines of code; others specifically exclude them. If asked to measure his program without comments, the programmer can only estimate. If he has the time and inclination, he may sample parts of his program to get proportions of comment lines to code lines; or he may guess. Either way, additional error is introduced into the measure.

Data specification statements are eliminated from size estimates for some models. As in the case of the comment lines, the proportion of specification statements may vary substantially from program to program. There is no ready way to count the number of specification statements in FORTRAN. By comparison, COBOL groups these together for easy access.

Compilers usually make it easy to share source code among programs. Often code is stored in libraries and automatically called in by the compiler at the time of complilation. Some compilers count the copied lines in addition to the rest of the program lines; other compilers count only the lines calling the library code; and still other compilers count both sets thereby producing two line counts. Unless the cost analyst has specified how copied code is to be counted, the responses among different programmers will not be consistent.

The number of object instructions associated with a program written in a higher order language has at least as many possible interpretations as the number of source statements.

When a program is written in a higher order language, the code produced by the compiler and executive support programs is of little

concern to the programmer unless the code is constrained by size or speed limitations. But even if the programmer is faced with these problems, normal practice is to work with the higher order language to satisfy the constraints. The tendency to work with the higher order language and the decreasing involvement of programmers with machine level operations mean that the information describing the attributes of the program in its executable form is not generated during compilation and link editing or, if it is, it is ignored or even not understood.

Given the inclination toward higher level language use, even if the definition of object program size were very precise, there are many programmers who would not know how to respond to it properly. Furthermore, these programmers are not very likely to be frank about their ignorance.

Probably the more common situation, however is that the definition of the number of object instructions is not precise. There are several ways that misinterpretations can occur.

Programs written in a higher order language (or in assembly language for that matter) go through a two-step process before they can be executed by the computer. The first step is under the control of the higher order language processor and it produces machine level code that needs other code before it can be executed. This code is sometimes referred to as a relocatable module. The relocatable module is processed by a link editor routine that produces an executable program. The executable program includes all the routines needed in addition to the relocatable module and it has an integrated addressing scheme. The executable module is the program that was needed to solve a given problem. But it can easily be several times the size of the sum of the relocatable modules that were actually written. Much of the executable code is taken from the executive library and performs standard mathematical functions or drives the computer system peripheral equipment. Furthermore, some computers copy the basic

library routines many times and insert them in the code; others share the common code among the modules without duplicating it. Therefore, the same definition of program size can elicit very different responses for a program written to the same specifications because of the way the computer executive software operates or the way the programmer directed the link editing process.

The computer memory occupied by the executable program contains not only the instructions but also areas reserved for program constants, data, and temporary input-output data storage. These areas in total indicate the amount of memory required to execute the programs. The available memory may be a constraint or otherwise specified by the user. Sometimes the size of the executable module in words of memory is used synonymously with number of instructions. These two measures may be substantially different.

The data storage area size may be very small compared with the memory area containing the executable instructions or it may be many times as large. Therefore, if the number of instructions is taken to be the words of memory occupied by the data and instructions, a large error may result.

Most computers do not incorporate instructions that are all the same size. The number of words of memory required to contain a single instruction may range from one to several. Some estimate of the average words of storage per instruction must be made. This average will depend on the distribution of the different length instructions in a program and would vary with different types of programs. A program that has a large percentage of input-output would have a longer average instruction than one in which logic and mathematical computations predominate.

In order to avoid introducing error into the program size definition it is necessary to know the following information:

- Higher Order Language
  - Whether the measure is statements or lines of code
  - Whether comments are to be included
  - How code copied from another source is to be counted
  - How data specification statements are to be treated
- Machine Level Language
  - How to count instructions
  - Whether data areas are to be included
  - Whether relocatable or executable code measures are wanted
  - How to describe the library or copied modules.

Some of the models give very explicit instructions on how to describe the code size measure. When such information is provided, we can describe the size input with confidence. Other models are vague about the definitions and we have had to make some assumptions about what measure is appropriate.

The purpose of this discussion has been to point out the errors that may be associated with the use of the size measurement in software cost estimating. We have been mindful of the points presented above both in collecting and interpreting the data used to evaluate the models for this study and in preparing inputs for the different models. We have read model descriptive materials and when possible we have contacted the authors when necessary to clarify definitions. When it was necessary to make assumptions about either the data or the inputs, we have described them. We have also prepared estimates of the errors associated with possible interpretations of size measures. Table 6 summarizes the size definitions found to be used by the models in this evaluation.

TABLE 6

SIZE DEFINITIONS

USED IN THE DIFFERENT MODELS

	AEROSPACE	BOEING	DOD MICRO	DOTY	FARR & ZAGORSKI	PRICE S	SLIM	TECOLOTE	WOLVERTON
HIGHER ORDER LANGUAGE[1]									
WRITTEN[2]	U	S[4]		L		S[6]			
EXECUTABLE							S		
DATA DEFINITION							S		
COMMENTS									
ADAPTED	U			L		S[6]			
EXECUTABLE							S		
DATA DEFINITION							S		
COMMENTS									
COPIED OR TRANSFERRED INTACT	U								
EXECUTABLE							S		
DATA DEFINITION							S		
COMMENTS									
MACHINE LEVEL LANGUAGE[3]								I	
WRITTEN	U								I
EXECUTABLE				W	I	I[4]			
DATA DEFINITION									
COMMENTS									
ADAPTED	U								I
EXECUTABLE				W	I	I[5]			
DATA DEFINITION									
COMMENTS									
COPIED OR TRANSFERRED INTACT	U				I				
EXECUTABLE									
DATA DEFINITION									
COMMENTS									

(1) Described as Lines (L), Statements (S), Unspecified (U)
(2) May be further specified as Delivered or Not Delivered
(3) Described as Number of Instructions (I) or Number of Words of Storage (W), Unspecified (U)
(4) Delivered Code Only
(5) Judgement necessary for consistent results
(6) Secondary input may be used to calculate primary input.

Development Effort Definitions. Many of the differences between the stated needs for estimating costs and the outputs of models are semantic. Models are directly or indirectly formed from experience and what is being predicted depends on what has been observed. Past data is seldom the product of uniform consistent definitions or is it even composed of the same elements. Furthermore, there is no common set of definitions that are accepted by all cost analysts. When we ask different cost accountants what is included in a data set under program design cost, we will get different answers. Even if the answers are in the same terms there is no guarantee they are consistently interpreted by its originators.

Our best approach is to consider the costs on a relatively high level of definition (total costs as opposed to subtotals). Then our main concern is scope. This we will address as carefully as possible. All this is said not to be negative, but to present an accurate picture of the conditions under which all cost estimating is done. We as an industry have established a certain value on historical cost data and this produces results of a certain quality. As long as the results obtained with this data are satisfactory, there is no reason to invest resources into making it better. We can tell if the presentation of the data is consistent with the Air Force life cycle definitions and the prescribed Work Breakdown Structure (WBS), but when totals are given we cannot always know if the elements specified are actually included in the model estimates.

Different elements that comprise development time:

1. Actual at-the-desk design, coding and testing hours. The physical direct effort required to produce the code.
2. The time charged to the project but including lost time or inefficiency (breaks, small routine administrative chores, etc.)
3. The time not usually charged to the project, but part of every job. It is understood that in order to realize the

direct effort described in 1., it is necessary to overstaff so that when these additional hours are lost, the proper net will result. This last category includes: sick leave, vacation, training, other scheduled lost time.

Depending on the method used by an organization to identify how personnel time is accounted for, different portions of items 2 and 3 will be associated with a given project.

For example, consider how two government organizations, one from the Air Force, the other from the Army account for non-project time.

AIR FORCE

	Hours per Month	
	Military	Civilian
Holidays	6.0	6.0
Leave	6.9	14.8
Medical	3.9	6.8
Education and Training	3.8	1.4
Social Actions, Organization Duties, etc.	9.4	1.0
	30.0	30.0

ARMY

Non-Project Training
Vacations
Excused Time
Holidays
Military Leave
Sickness
Non-Project Meetings
Special Assignments
Assignments to Other Projects

} 44 Hours per Month

There are some measurements that are not possible to interpret with confidence even after careful analysis of the sources and contacts with the model developers. Questions would have to be asked of each respondent or originator of an item of data. We can only list the possible errors and speculate about what implications the probability of their existence has for the interpretation of the results.

4.2 MODEL TYPES

Our objective is to obtain insights into the relationships between types of models and their prediction qualities. Therefore, it is desirable to establish some method of characterizing the models that describes their approach to making estimates.

We have selected a scheme that classifies model structures into three types:

- Regression
- Heuristic
- Phenomenological

The following discussion explains why these classifications were selected and how they are related to the development of more accurate models.

Classification of the models by type is equivalent to forming a hypothesis about the structural characteristics that affect accuracy. Selecting only a few types limits the consideration to the major structural attributes. This is more likely to provide statistically meaningful results.

At the detailed level, each model is unique. It makes its estimates using different parameters and procedures. If it is better or worse than another model, we don't know to which of the differences between the models

to attribute the performance. On the other hand, if we hypothesize that the models fall into certain catagories and if the categorizations are related at least intuitively to accuracy, then the measures of prediction performnace can be associated with the selected attributes and model development can be directed toward structures with the desirable attributes. The process decreases the number of model characteristics that are considered to contribute to estimating performance. If the results indicate no relationship between the categories and prediction accuracy, then the hypotheses can be restated and the analysis repeated.

Top down and bottom up are not sufficient to describe models. For example, SLIM estimates the development cost (which is not the same development cost estimated by the Boeing, Doty, or DoD Micro models) for the entire system then extends the scope of the estimate to include the Requirements and Specifications phases. The Design and Coding, Integration and Testing, and Installation phases are fixed portions of the Development Phase. Therefore SLIM at least in one of its operating modes, is a top level estimate constructed from individual subsystem size estimates. That initial estimate, however, includes only a part of the life cycle effort. The model derives higher and lower level phase elements from the core estimate.

The Dod Micro Procedure describes the system at the function level and assembles the weighted components into a net development effort for the entire system. The net development is extended to include indirect effort and to create a total system development estimate. This is then decomposed into phases at the system level.

These two examples illustrate the problem of trying to fit models to the simple descriptors "top-down" and "bottom-up." It is necessary to indicate what the top and bottom refer to. When is a system described at the top? How does the scope of the life cycle included in the estimate compare with the level of detail in the system description?

We have tried to develop a method of describing system structures that considers these questions. By following a line of reasoning that utilizes a two-dimensional description of software life cycle effort it is possible to address the life cycle scope and system level of detail independently. This is an important distinction that has not been explicitly addressed by most model developers (Wolverton and PRICE S are two exceptions). Proceeding in this manner it is possible not only to describe models more accurately, but it provides an insight into the way the different models proceed from inputs to outputs that is the basis for categorizing the structures. Hopefully the way models proceed: the method of making the initial estimate, the method for extending or detailing the scope and the way of developing detail can be associated with prediction performance.

As a first attempt, the categorizations are very broad. They concentrate on the method of making the initial estimate. They also indicate the general method of making the subsequent estimates. If we look at the nine models included in this study, we can see three distinct ways of making estimates. These are described in the following paragraphs.

The Regression Type of Model Structure. One class of model structures reflects a design based on the selection of the life cycle element of interest (e.g. life cycle effort, development effort, or coding effort) and a hypothesized relationship between the element and a number of selected inputs. The parameters of the hypothesized relationship are obtained by regression and the model becomes a single cost estimating relationship which is treated as valid for whatever population is believed by the creator and the user to be represented by the data used to calculate the parameters. The data may be stratified in some way thereby producing a set of estimating equations with each member of the set applicable for the estimating situation described by the stratification parameter.

We have termed models structures this type "Regression Models" and they include the Aerospace, Doty, Farr and Zagorski, and Telecote models. The scopes and levels of these models differ, but they share the attribute of a single estimating relationship derived by linear or log-linear regression using various inputs. This is believed to represent a recognizable approach to model construction and it is an objective of this evaluation to learn if the approach produces more accurate estimates than the others.

The Heuristic Structure. Looking again at the models under investigation we can identify another approach to making estimates.

If we examine the Boeing model, for example, we see that the system is divided into groups of code that are characterized by type. Each type has an associated productivity. These values have been obtained from historical data by visual curve fitting, regression or by subjective assignment. The application of judgement both in the creation of the procedure and in the establishment of parameters is typical of this class of model structures. A system level estimate is calculated by summing the effort for the different code groups. The system level estimate includes the entire development cycle. This is divided into phases using fixed ratios. Again, the ratios may be obtained by objective or subjective means. The phases are adjusted to account for development or system-related factors that are believed to be phase dependent. The adjusted phase values are added together to produce an adjusted development effort.

PRICE S, calculates the cost of the Engineering Design life cycle element from values of system size, resource, and complexity. Adjustments are made to this value to account for development time, technology, and other factors. The adjusted value is used to calculate the other elements of the system development cost by means of a cascading technique. The final time distribution of system cost is subjected to further adjustments to obtain a predetermined time phasing of the development cost.

The Boeing and PRICE S models along with the DoD Micro Procedure and the Wolverton model have been termed "heuristic." The dominant characteristic of the Heuristic models is their freedom from any single mathematical formulation. This distinguishes them from the cost estimating relationship that is the hallmark of the Regression type of structure. Heuristic models usually combine a number of different estimating techniques. The calculation of the estimate usually flows through a series of estimates and adjustments. The selection of the individual steps, the cost elements treated in the steps and the method of determining the adjustment parameters differ significantly among the models. However, as an approach to making software cost estimates it is describable and distinguishable from the Regression approach. Measuring the differences in estimating accuracy between the two methods would give considerable guidance for future model development.

The Heuristic model structure combines observation and interpretation with supposition. It is the formal representation of the subjective process of applying experience. Relationships among variables are stated without justification (e.g. cost per pound decreases with increasing size, development effort is related to number of file formats, the number of instructions per month depends on the type of code, etc.). Then subjective, semi-empirical, or empirical adjustments are made to the base estimate. In some of the models included in this evaluation this process is extremely complicated.

The advantage of the Heuristic structure is that it does not have to wait for the establishment of formal relationships describing how the cost-driving variables are related. The process of model development proceeds intuitively. As situations are encountered where the model fails to perform acceptably, an adjustment or addition is made and the process continues. Over a period of time a given model can become very effective in a stable predicting environment. It becomes the repository for the collected experience and insights of the model designers and users.

The Phenomenological Model. One model in the evaluation group, SLIM is unique in that it incorporates a concept that is explainable in terms of a basic phenomenon that is not limited to the mechanics of software development. The relationship can be derived in terms of the rate of solving problems.

SLIM uses the Rayleigh-Norden function [36] [37] [38] to describe the time distribution of effort during the software life cycle. It has been shown [36] that this function represents the time distribution of effort required to solve a given number of problems under the assumption of a constant learning rate. The ability to describe observed processes in terms of elementary phenomena is characteristic of the more mature sciences. It allows complex relationships to be explained by interactions among elementary functions. These functions are verifiable by controlled experiments.

Although it may be argued that SLIM incorporates too many empirical adjustments to be a purely phenomenological model, it nevertheless is the only model to use an observed basic relationship to make estimates.

The difference between a Phenomenological model and a model based on one or more hypothesized relationships which could be used in either a Regression of Heuristic structure depends on the source of the hypotheses. If the hypotheses are motivated by tendencies observed among the variables describing software and its development resources, and if the hypotheses are describable only within the context in which they are used, then the resulting model structure is not phenomenological. The phenomenological model must incorporate ideas or processes that can be observed and measured independently of the software development process. The justification can be derived after it was observed in a software context and a model may become phenomenological after it was previously classified otherwise if the qualifying condition is satisfied.

Given the present situation, where no generally accepted statement exists of the elements of the software development process, the true phenomenological model is mostly an ideal. But the search for basic understanding and description must be made if we are to obtain real improvements in prediction quality. After a while a heuristic model may collapse of its own weight as it tries to adapt to each new experience. The phenomenological model ultimately promises the simplicity of representation that characterizes scientific laws.

The development of phenomenological models requires the explanation of basic processes. Some of these processes are being investigated. Halstead [39], for example, has explored the relationships between algorithms and the effort needed to code them. Other researchers have identified phenomena related to the development of systems such as the law of increasing entropy [40], life cycle phase interrelationships [41], ripple effect [42], and others. These elements will undoubtedly contribute to the establishment of new models based on elementary phenomena.

Describing the Estimating Process. The definition of the model type classifications is intended to establish general approaches to estimating that are associated with greater or less prediction accuracy. Within each general type of structure it is necessary to describe more detailed aspects of the estimating process.

The estimating procedure is described according to the cost element that is estimated first and the method used to make it and then the method used to obtain subsequent estimates.

The cost element used for the initial estimate has an important bearing on the level of aggregation that is associated with the greatest accuracy. There may be certain combinations of the level of the first estimate and different methods for making it that have implications for accuracy.

Having identified the best way to obtain the initial estimate, it should be useful to investigate how subsequent estimates are either expanded in scope to obtain a synthesis of the entire life cycle or decomposed to allocate portions of it.

The classifications used for describing the estimating procedure are as follows:

Level of Initial Estimate

- System Total Development
- System Analysis
- System Design
- System Coding
- System Test
- CPCI Total Development
- CPCI Analysis
- CPCI Design
- CPCI Coding
- CPCI Test

Method of Making Initial Estimate

- Single Parameter
- Multi Parameter

Method of Making Subsequent Estimates

- Cost Estimating Relationship
- Ratios
- No Further Decomposition

4.3 TEST DATA SETS

The characteristics of each test data set are described in Appendix D, *Data Preparation*. This section describes the environments from which the data were obtained and tells how they were obtained.

The data used to evaluate model estimating performance was compiled from three sources:

- U.S. Air Force Data Systems Design Center (USAF/DSDC)
- Goddard Spaceflight Center, Software Engineering Laboratory (GSFC/SEL)
- The system development center of a large corporation

The data sets from the first and third sources were obtained by GRC from the developing organizations using questionnaires and other devices that will be described later. The second data set was given to us by Prof. V. R Basili of the Computer Science Department, University of Maryland. The University operates the Software Engineering Laboratory under a grant from the National Aeronautics and Space Administration (NASA).

Air Force Data Systems Design Center. The DSDC develops large, standard data systems for Air Force use world-wide. These are data management systems such as payroll, logistics, and personnel applications. The programs are written in COBOL and are developed and maintained at a single site. Under two separate Air Force contracts, (Electronic Systems Division, 1975 [43] and Sacremento Air Logistics Center, 1978 [44]) GRC collected data describing data system development hours, system characteristics and personnel data. From these data 17 projects were selected for use in this evaluation.

Data describing the hours charged by individuals to each project were obtained from the PARMIS history files. PARMIS was a project status reporting system used at the Design Center during the years 1971 through 1978. Hours were reported by project staff members on a weekly basis. The hours were identified according to project and activity.

GRC wrote a computer program that tabulates the hours for each system according to a standard set of life cycle phases and activities.

A questionnaire was given to project leaders who provided information describing the systems and the experience of the personnel.

The characteristics of the programs were obtained from two different sources. The command-level systems, which are implemented on Honeywell H-6000 computers, were processed by a program called the Program Profile System (PPS), which was developed at the Design Center. The PPS analyzes the source code and tabulates the number of lines, statements, record descriptions, etc. The base-level systems, which are implemented on Burroughs B-3500 equipment were described in the materials prepared by the Air Force as part of the request for proposals to replace the base-level systems.

Using these sources and with considerable help from the personnel at the Design Center, a rather complete and reliable set of data have been compiled describing the systems.

Goddard Spaceflight Center, Software Engineering Laboratory. The SEL is responsible for maintaining a high quality data base describing the software development experience at the Goddard Spaceflight Center. A full-time staff collects and analyzes data describing the system attributes, development methodologies, and resource expenditures. The software operates on large ground-based computers in support of satellite operations. The primary language is FORTRAN and much of the code operates under a time constraint.

The data used for the evaluation of estimating accuracy represent seven systems. Two of the systems are partitioned into ten subsystems.

Commercial Data. GRC has an arrangement with a large corporation whereby the two companies exchange information describing software development experience. The data used in this study was provided by the company's central system development facility. The applications are

data management systems written in COBOL and two data base management languages. The facility utilizes modern programming practices and provides the programmers with online programming and debugging capability.

Data describing eleven systems were tabulated from company records onto a modified version of the questionnaire used to capture the Air Force Data System Design Center data.

4.4 MISSING DATA

The models selected for evaluation represent a range of different estimating methods. They were constructed in many environments and as a result they include as a group many different input and output variables. It was demonstrated in the last section how two of the most common model variables, size and effort, can be represented by over a dozen different definitions. Add to these differences, the number of types of variables that may be used in model construction and it is easy to see how the requirements for test data variables becomes very large. The nine models in the test group require more than thirty variables to define their inputs and outputs, and that does not include the minor variations that exist among the variables.

It was not possible to obtain test data that includes all the model variables and their variations. The compatibility between data availability and that needed to execute and compare model outputs is represented in Figure 5.

The Roman numerals identify the inputs and outputs contained in a given data set. A data set may contain some of the inputs for each of the models (Greek letters) and some of the outputs, but all the models could not be executed and compared using the same data.

Our approach was to execute each model according to the outputs it provides and the inputs needed to make them. If a model is designed to predict design, code, and test effort, then the test data were adjusted to describe these portions of the life cycle.

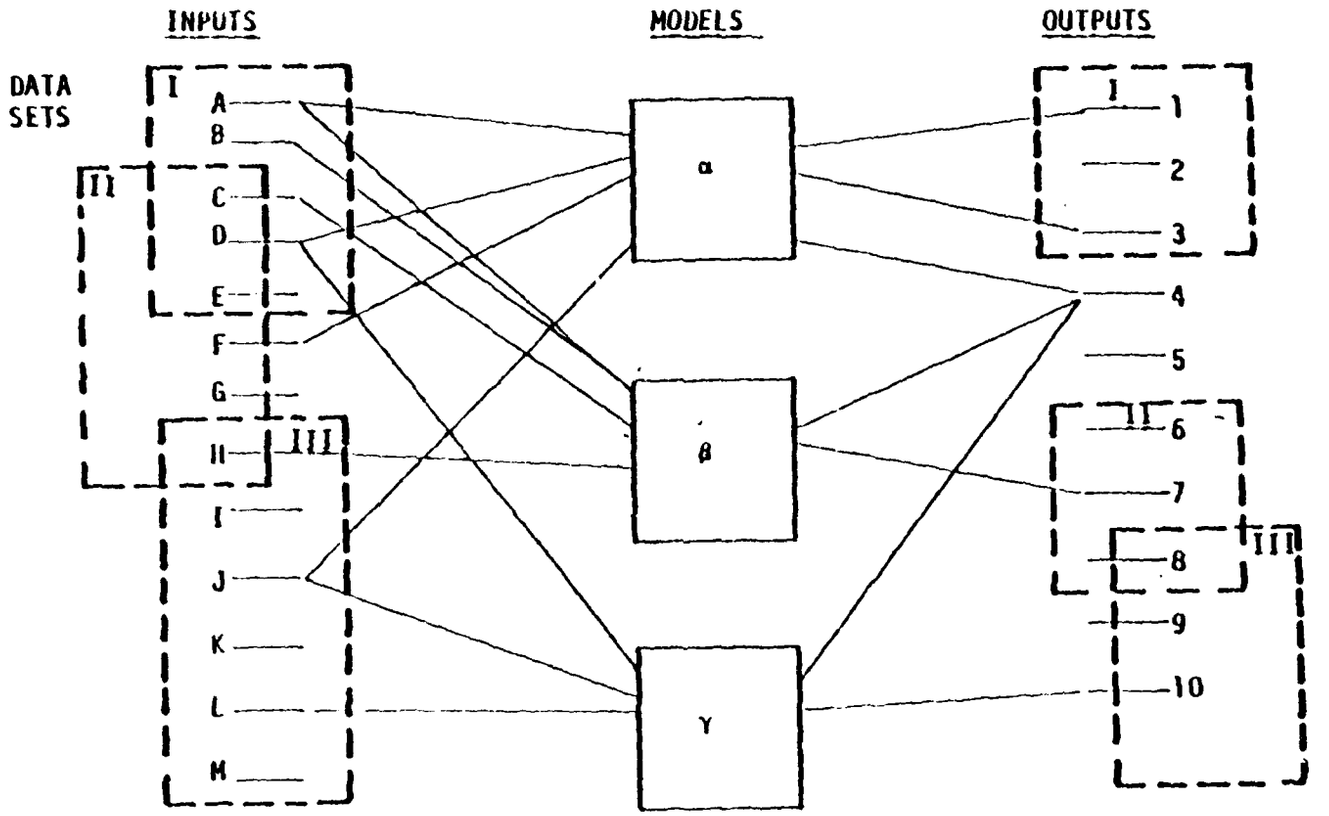


Figure 5. Problems in Compatibility Between Data Sets and Model Variables

Missing inputs were obtained using estimating relationships constructed from the other inputs. For example, if a model called for a number of object instructions and a given data set contained only lines of source code, a relationship was developed from other available data that could be used to predict number of object instructions from lines of source code.

Whenever it was necessary to estimate missing input data, both the expected value and its variance were estimated. The variance is used to indicate the effect of the uncertainty in the estimate in the evaluation of prediction accuracy.

The input data for each of the models are presented in Appendix D, Data Preparation. The effect of missing data on the evaluation are discussed in Section 6.0, Analysis of Results.

5 RESULTS

The cost model evaluation criteria (Section 3) were designed to consider two aspects of the state of the art in software cost model construction:

- The satisfaction of US Air Force needs for software cost information (Section 3.1); and
- The satisfaction of US Air Force needs for cost estimating accuracy (Section 3.2).

The objective of the evaluation of the first aspect is to measure how well existing models satisfy Air Force information needs with regard to software cost items and their levels of detail.

The evaluation of the second aspect, accuracy, relates estimating accuracy to model structures. The objective is to identify ways of making estimates that can be demonstrated to produce greater accuracy. The objective does not include a ranking of the existing cost models or any general statements of their individual estimating accuracies. The results will be used to design a research program for improving the accuracy of software cost models.

A comparison of the outputs of the models under investigation with the Air Force estimating needs indicates the following:

- The supporting materials for most of the models do not clearly state the elements included in their estimates and are not precise about their definitions.
- The existing models are better able to satisfy information needs early in the acquisition life cycle.
- None of the models included in this study fully satisfy the Air Force need for information either with regard to scope or detail.

- The models tend to be phase oriented and do not properly describe activities that cross phase boundaries. This precludes obtaining data compatible with both management planning (phase related) and product cost (WBS).
- Although most of the models use the summation of program or module sizes to make their cost estimates, only one model studied provides for keeping track of the cost on a component basis and accounts for the cost of system integration. None of the models provide for all four levels of system definition called for in the Work Breakdown Structure (Ref. Appendix B).

Based on the relative root mean square error measure of performance:

- Recalibration* is the primary factor contributing to the differences in estimating performance among the models tested.
- The contribution of model structure* to estimating accuracy is not significant when the models have been calibrated to the development environment*.
- The development environment significantly affects the relative performances of the models tested.
- The effect of development environment on estimating performance precludes the possibility of obtaining generally applicable measures of model performance without applying additional controls.
- Models that do not use size as an input may perform as well as those that do.
- The average RMS Error for all tested models is unacceptably large for Air Force estimating purposes.
- The use of models that are not calibrated to a given development environment can lead to very large estimating errors.
- The best performance obtained by any group of the models tested is not adequate for Air Force needs.

The detailed presentation of the results of the evaluation of the software models is in two parts. The first part compares the outputs or

* Definitions of these terms are given in Section 1.5.

estimates produced by the models with the needs associated with the major weapon system development process. The second part shows the prediction accuracy of the different models.

5.1 COMPLIANCE WITH AIR FORCE COST INFORMATION NEEDS

The description of each of the models was studied to learn exactly what the outputs represent. The Air Force estimating needs (Figure 4) require information that includes certain cost components (administrative expenses, non-delivered software expenses, overhead, holidays, etc.) presented at specific levels of the Work Breakdown Structure (Appendix B. Table 5).

The following paragraphs describe the outputs of the models in terms of the needed information. In many cases it was not possible to determine whether the information being sought is included in the model estimates. Model descriptions are often vague about the details of the outputs. Sometimes it was necessary to acquire a detailed knowledge of the original data used to construct the model in order to identify the elements. It is likely that precise answers to some of the questions of definition are unknown to anyone save the individuals who originally recorded the data. In many cases, data were not recorded consistently.

Our attempts to obtain precise definitions of model estimates and to identify the included cost elements indicates that:

- The supporting materials for most of the models do not clearly state the elements included in their estimates and are not precise about their definitions.

Our approach was to read the model descriptions, check the assertions with the original data source when possible, ask questions of the model creators when they were available and finally to draw on our own experience. Hopefully, the results are valid descriptions of the model outputs.

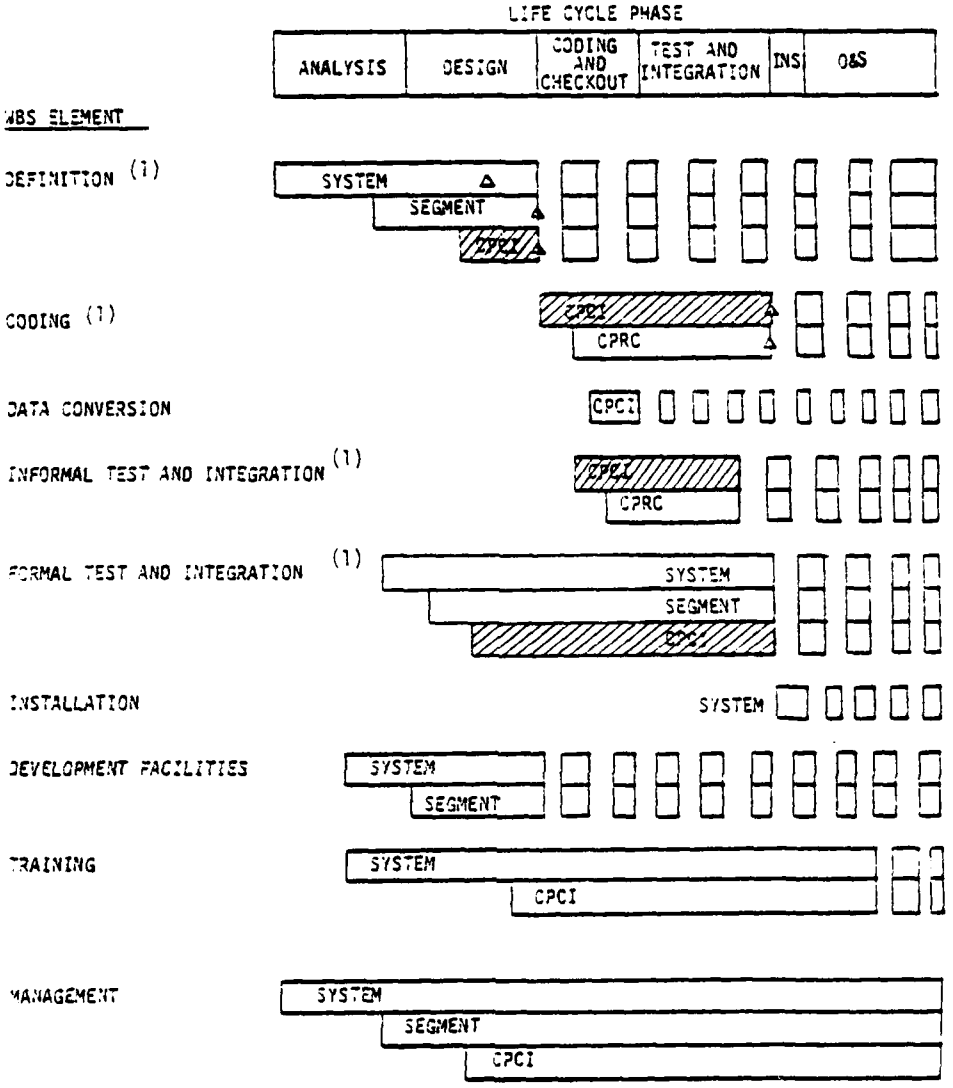
Notice that the figures (Figures 6 through 14) differentiate between output items that are explicitly presented (shaded segments) and those that are included as parts of other values (cross-hatched segments). The numbers in parentheses indicate how some of the items are grouped for presentation by the model. A model that completely satisfied all Air Force cost information needs would have all the segments shaded.

Aerospace Corporation (Figure 6). The presentation is typical of the regression models. System level estimates of cost or effort are obtained by fitting linear or log-linear functions to historical data. The elements believed to be represented in the total are indicated by the cross-hatching. However, since the historical data often comes from several sources, the components are often not the same. Data quality is usually a problem so little confidence can be given to any description of the elements included.

The figures show only those data items explicitly mentioned in the model descriptions. Therefore, items such as data conversion and installation may actually be included in the Aerospace Model estimates depending on the data collection and tabulation practices governing the historical data from which the model is derived. The model does not provide any breakdown of the system level estimate.

Boeing Computer Services (Figure 7). This model divides the system level estimate into life cycle phases. The process is similar to that used in a formal manner by PRICE S and informally or subjectively by the Wolverton model. In all three models the effort distributed among the phases is not in fixed proportions. In the case of the Boeing model adjustment factors reflecting environmental characteristics are phase-dependent. Therefore, the recombination of the phases makes a total that is different from the original life cycle estimate. PRICE S allows weighting factors to be applied to the distribution among phases and Wolverton allows the estimates to be changed to accomodate staffing and schedule constraints. All the other models that decompose the total effort into phases do so with fixed ratios.

AEROSPACE CORPORATION



△ BASELINE ESTABLISHED

Explicitly presented by the model.

Included, but presented as part of a higher level.

(1) Included in Development Cost

Figure 6 Comparison Between Estimating Requirements and Model Outputs - Aerospace Corporation

BOEING COMPUTER SERVICES

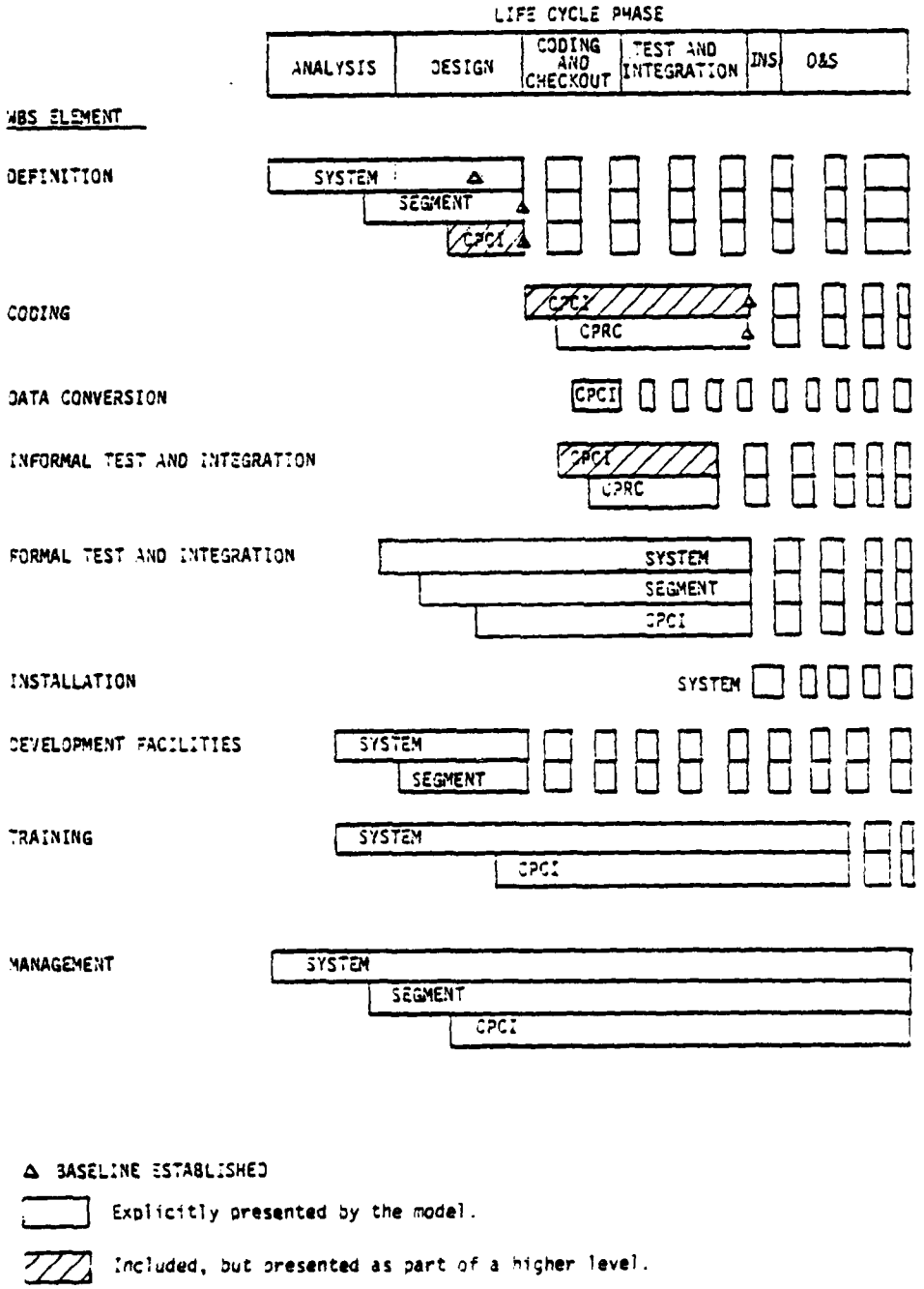


Figure 7 Comparison Between Estimating Requirements and Model Outputs - Boeing Computer Service

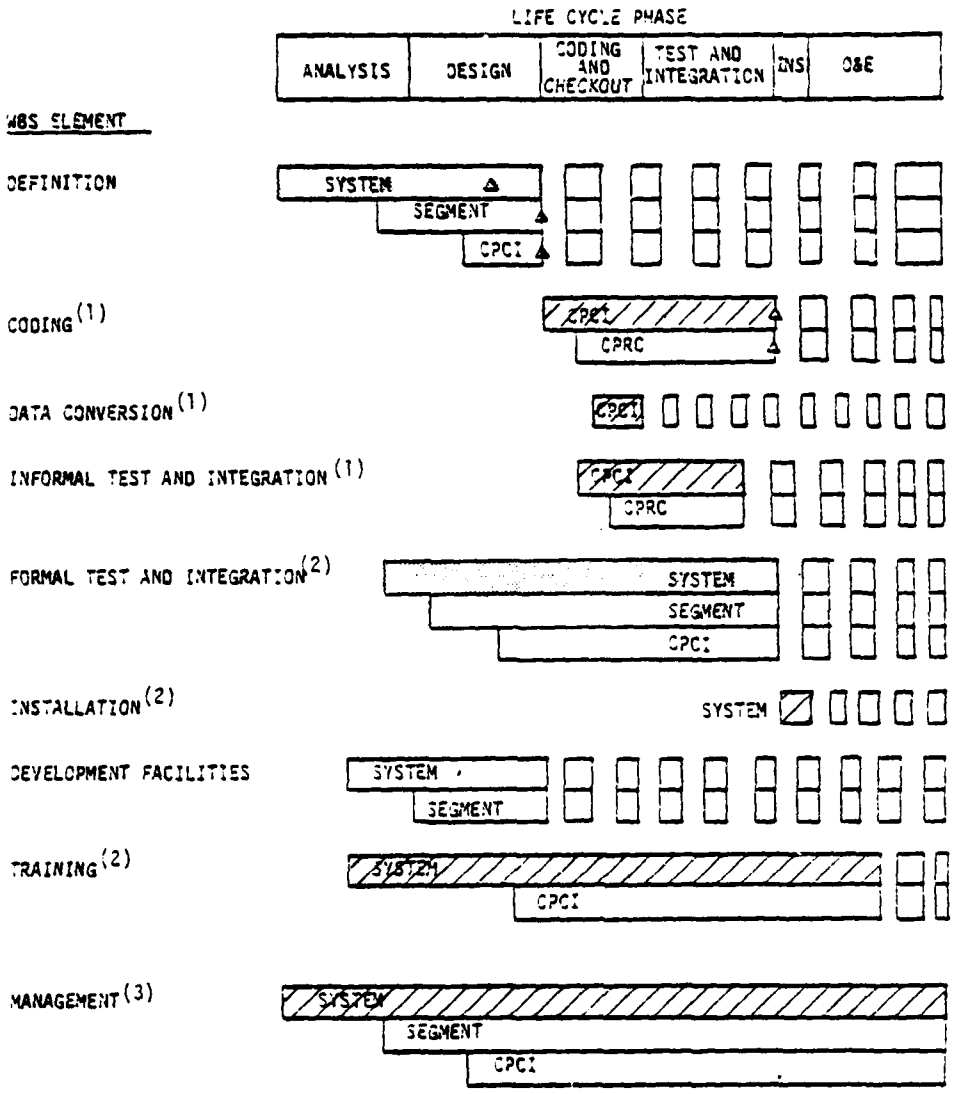
Notice that while the Boeing model includes the coding effort, only the portion that occurs during the Coding and Checkout Phase would be specifically identified. Any coding done during the Test and Integration Phase would be represented as Test and Integration effort. A similar misrepresentation occurs for design that is done after the end of the Design Phase. This problem exists in all the models. The proper identification of the elements in the work breakdown structure according to their proper occurrence in the life cycle will not be possible until considerably more effort is applied to the proper identification and tabulation of data describing the software development process.

DoD Micro Estimating Procedure (Figure 8). A system level estimate is obtained by a weighted count of program functions. The estimate is allocated to the life cycle phases using a fixed distribution. Training, management and other indirect activities are included as a multiplier of the direct effort.

Doty Associates; and Farr and Zagorski (Figures 9 and 10). These two models were derived from the same data definitions. The data used in the Farr and Zagorski model was a subset of that used in the Doty model. The estimates are made at the program level. The included activities begin with the detailed program designed and extend to release of the programs to integration and system testing. The Doty model includes an estimate of development time.

PRICE S (Figure 11). By properly choosing weighting factors for the three life cycle phases and five activities presented by this model, it is possible to obtain many elements of the desired cost information. PRICE S is unique among the models evaluated in that it allows subsystem level definitions to be explicitly presented for all of its life cycle and activity elements. Adjustments to account for the additional effort needed to integrate the subsystem into the system may be specified by the user.

DoD MICRO-PROCEDURE



Δ BASELINE ESTABLISHED

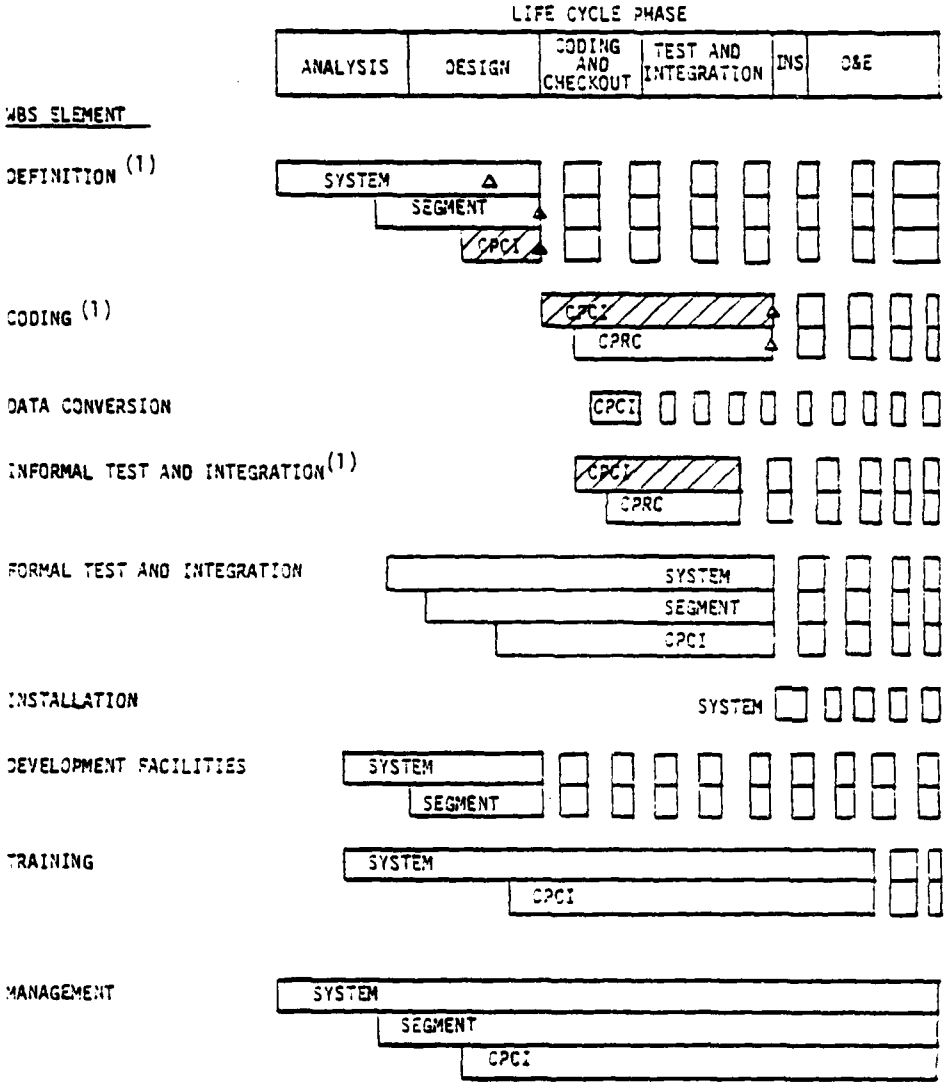
Explicitly presented by the model.

Included, but presented as part of a higher level.

(1) Combined into Programming
(2) Combined into Testing
(3) Included in all elements

Figure 8 Comparison Between Estimating Requirements and Model Outputs - DoD Micro-Procedure

DOTY ASSOCIATES



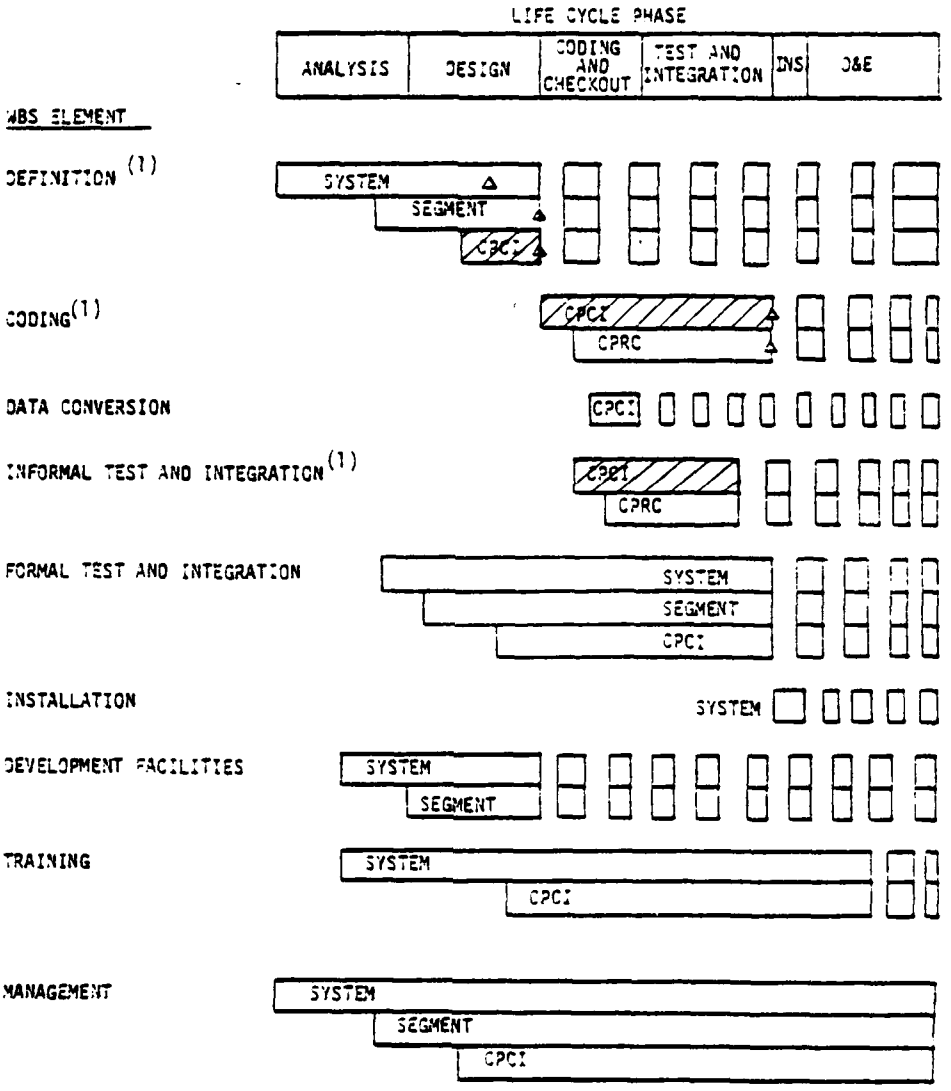
Δ BASELINE ESTABLISHED

Explicitly presented by the model.

Included, but presented as part of a higher level.

(1) Combined into Development Effort.

Figure 9 Comparison Between Estimating Requirements and Model Outputs - Doty Associates



Δ BASELINE ESTABLISHED

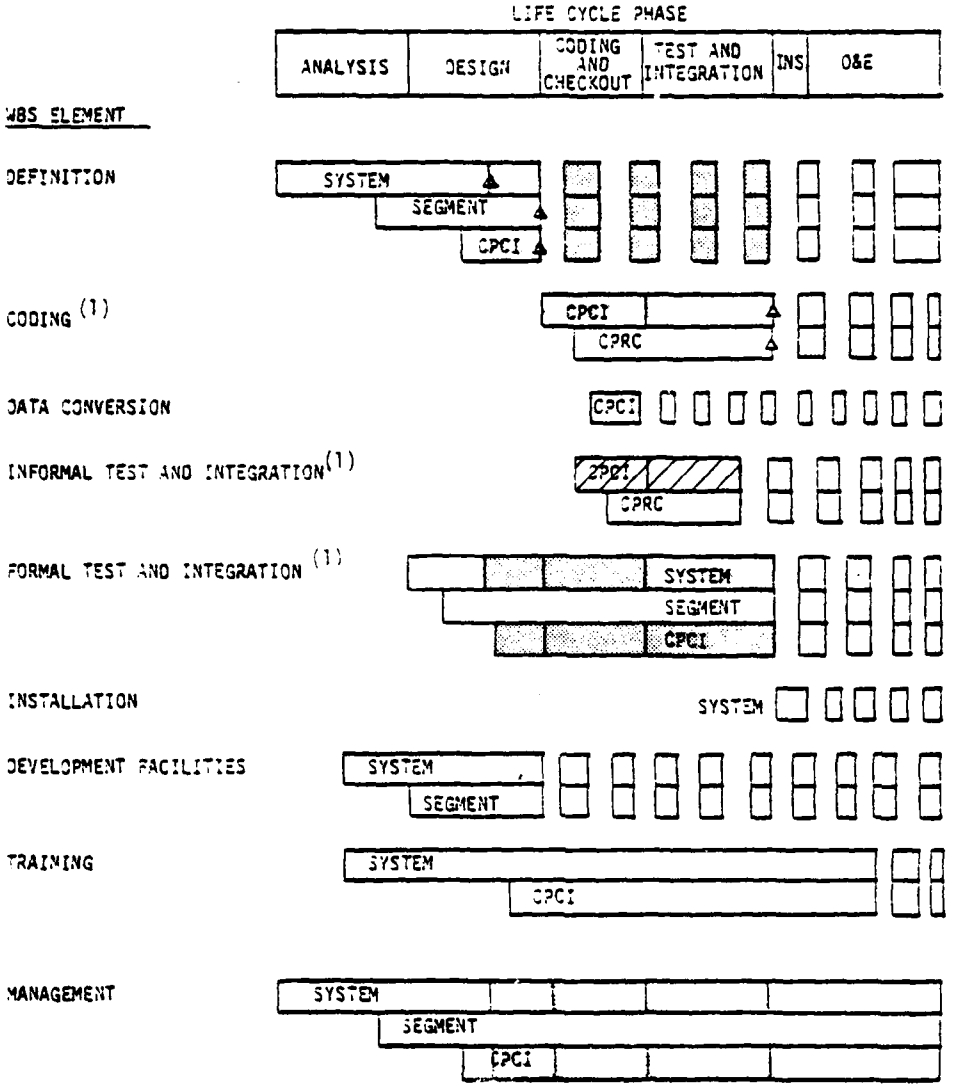
Explicitly presented by the model.

Included, but presented as part of a higher level.

(1) Combined into Development Effort.

Figure 10 Comparison Between Estimating Requirements and Model Outputs - Farr & Zagorski

PRICE S



△ BASELINE ESTABLISHED

Explicitly presented by the model.

Included, but presented as part of a higher level.

(1) Combined into Programming

Figure 11 Comparison Between Estimating Requirements and Model Outputs - PRICE S

The estimating life cycle begins with the detailed system design; that is, the definition of the allocated system functions. The phases overlap in time. The schedule is given.

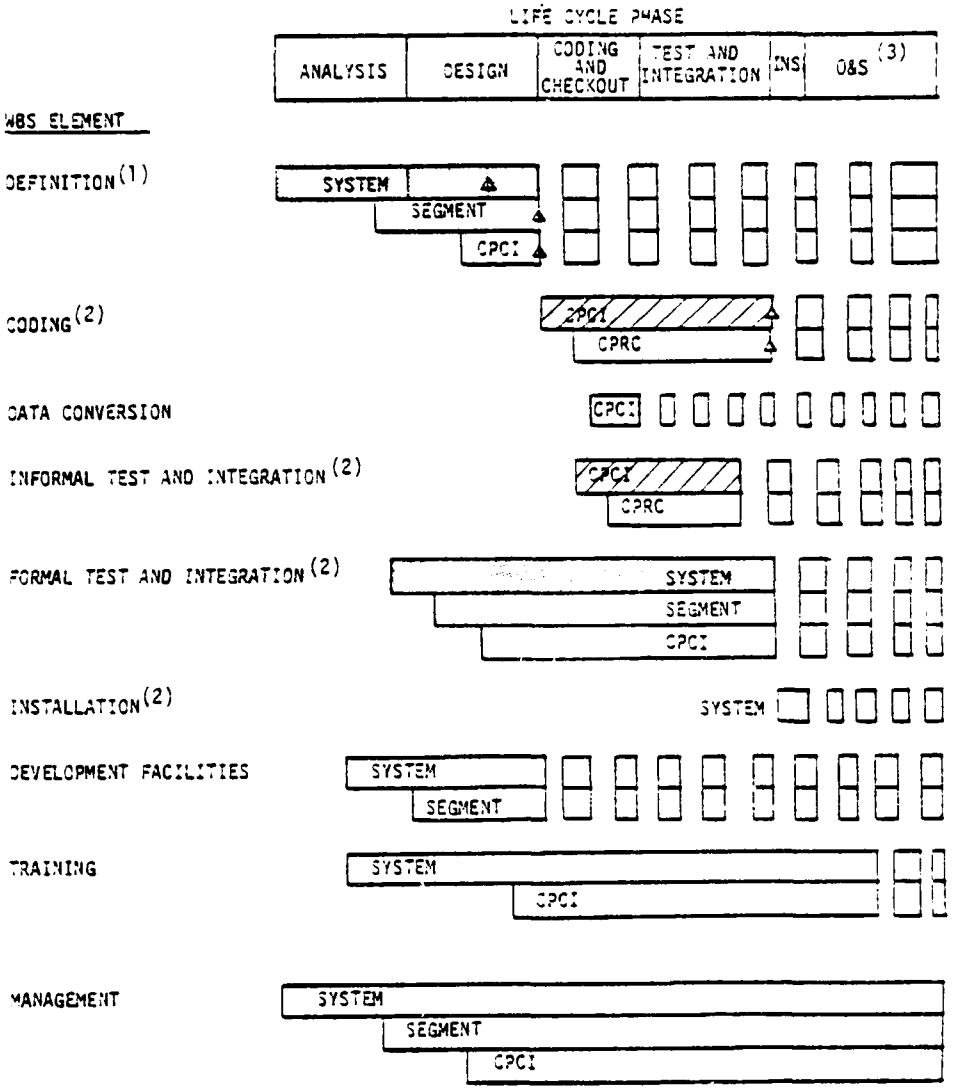
SLIM (Figure 12). SLIM produces a primary estimate of the development cost at the system level. It provides an optional "front-end" estimate that includes the Analysis and Design phases. The Operations and Support cost can be obtained from another option. Additional options provide estimates for computer hours and documentation. The life cycle components are described as overlapping and fixed in relative size. Milestone events describe the beginnings and ends of the phases.

Tecolote Research (Figure 13). A single system level estimate is produced by the model. No allocation of effort among the phases is given.

Wolverton (Figure 14). A very detailed matrix allocates the system development cost into seven phases each composed of up to 25 activities. Therefore, a complete description of phases and activities is obtained at the system level.

Model Compliance with Cost Estimating Situations. In Section 2, the Air Force need for software life cycle cost data was described in terms of five cost estimating situations. The imprecise nature of the data requirements and the dissimilarities in the outputs of the models preclude the creation of a checklist of defined data items. Therefore, the comparison of the model outputs with the needs must be partly subjective.

The comparison of model outputs with estimating needs is made in two dimensions: scope and detail. The scope of the estimating needs describes the cost elements associated with the life cycle phases. The scope of a model's outputs may be limited, for example, to the Coding and Checkout and Test and Integration Phases. The output detail describes the extent to which cost elements in each phase represent the system its components and its associated elements such as Facilities and Training.



△ BASELINE ESTABLISHED

Explicitly presented by the model.

Included, but presented as part of a higher level.

(1) Analysis is Feasibility Study; system level design is Functional Design; detailed design is included in Development Effort.

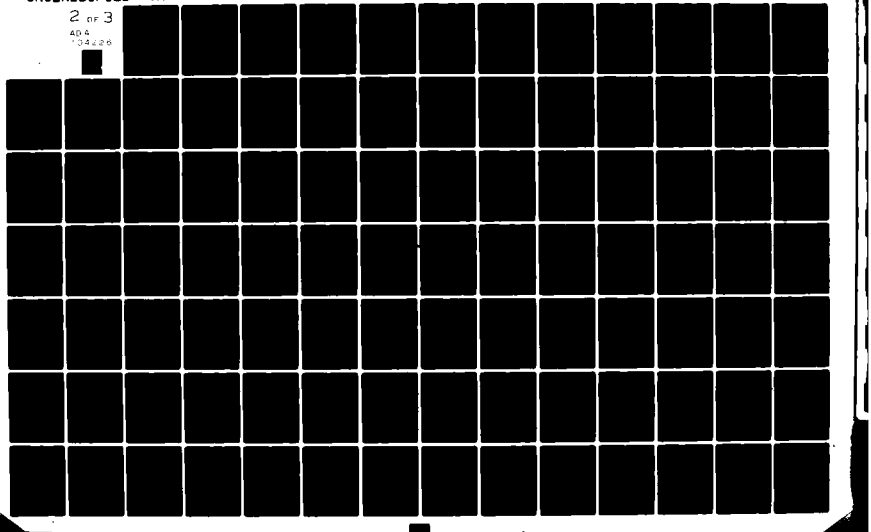
(2) Included in Development Effort.

(3) A maintenance cost and effort can be obtained from the Life Cycle option.

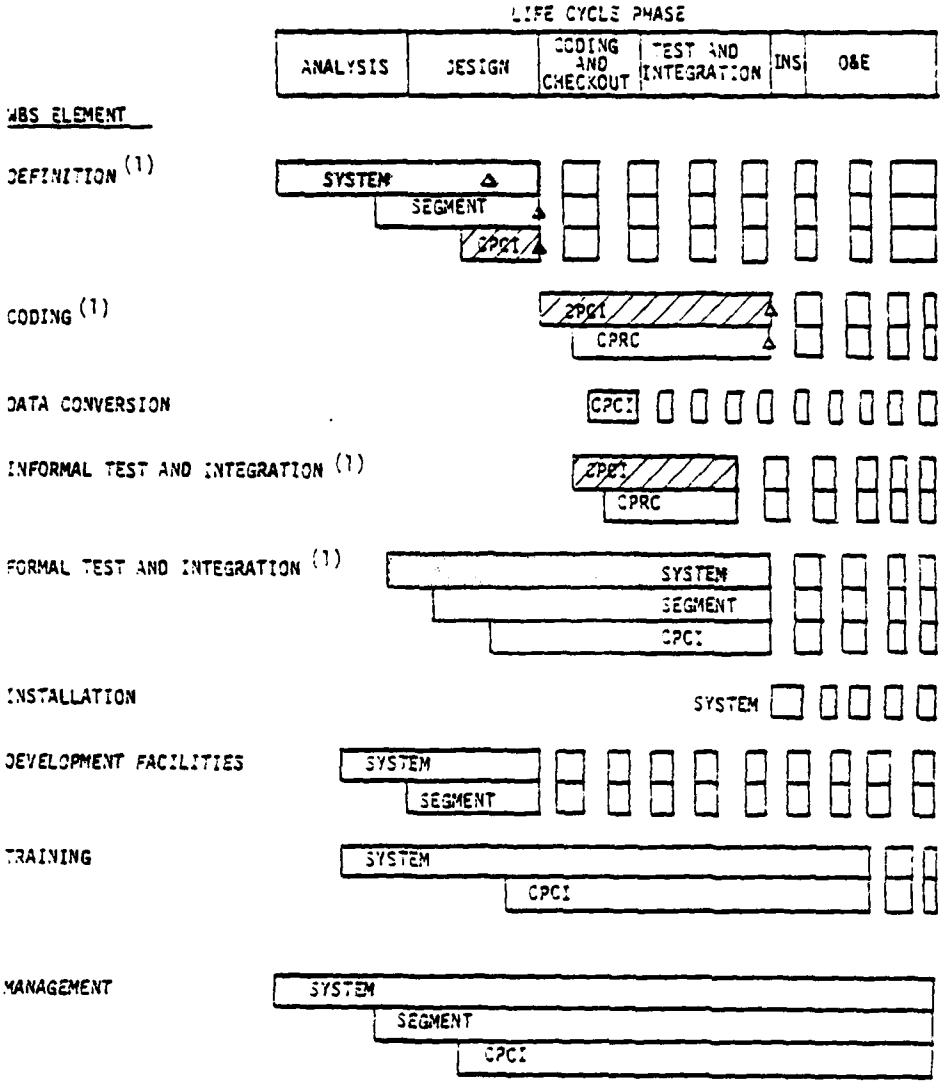
Figure 12 Comparison Between Estimating Requirements and Model Outputs - SLIM

AD-A104 226 GENERAL RESEARCH CORP HUNTSVILLE AL F/6 9/2
AN EVALUATION OF SOFTWARE COST ESTIMATING MODELS.(U)
JUN 81 R THIBODEAU F30602-79-C-0244
UNCLASSIFIED GRC-CR-1-940 RADC-TR-81-144 NL

2 OF 3
AD A
104226



TECOLOTE



Δ BASELINE ESTABLISHED

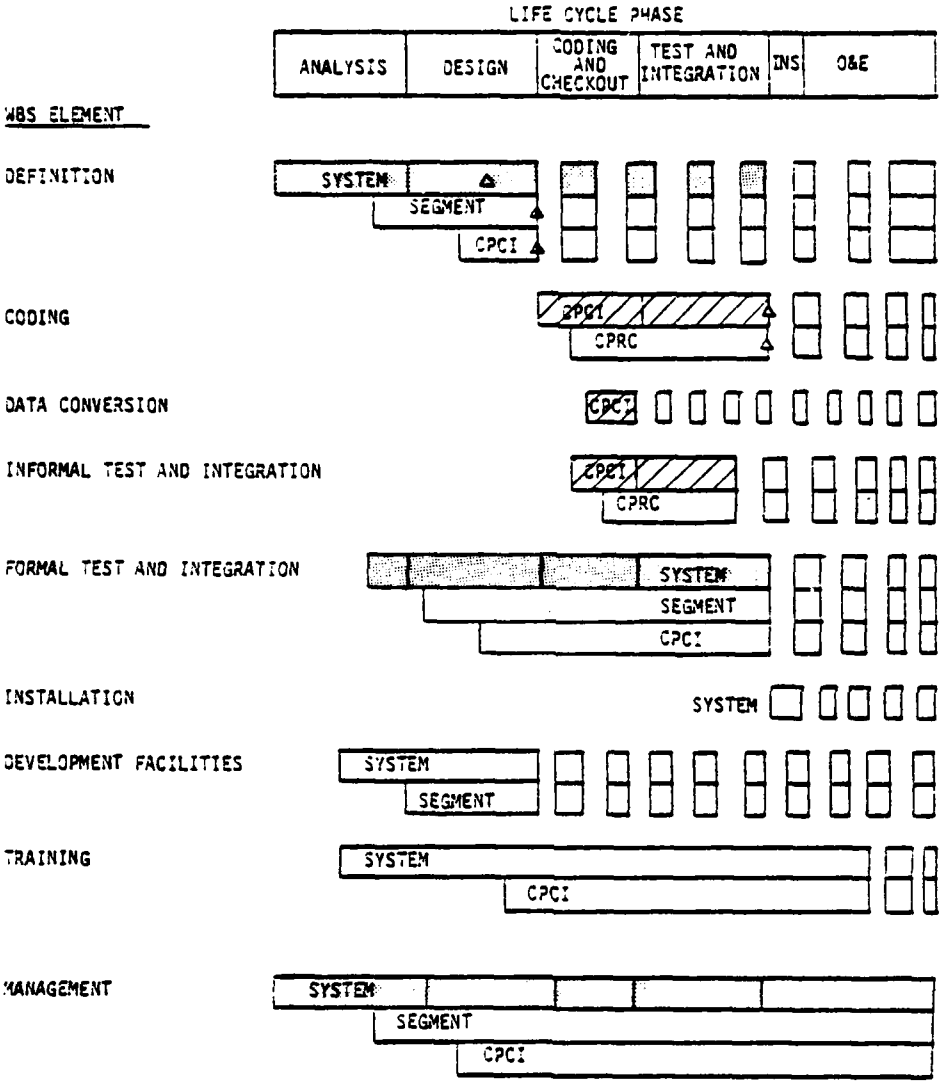
Explicitly presented by the model.

Included, but presented as part of a higher level.

(1) Included in Development Cost

Figure 13 Comparison Between Estimating Requirements and Model Outputs - Tecolote

WOLVERTON



△ BASELINE ESTABLISHED

Explicitly presented by the model.

Included, but presented as part of a higher level.

Figure 14 Comparison Between Estimating Requirements and Model Outputs - Wolverton

Table 7 is a summary comparison of the model outputs with the needs described in Section 3. A liberal interpretation of compliance was exercised in each case. A model was given credit in terms of scope if it addressed the required phases regardless of the possibility of differences in definition between the model and the standard. The same criterion was used to describe compliance with the needed level of detail.

The degree of model output compliance with each stated need is described using a scale of 1 to 5, where 5 indicates nominal compliance. In general a model was given one point for including each of the five major phases in its estimates. The Installation Phase was not included in the scoring. The detail scale was determined less objectively and depended on the analyst's view of the extent to which the model allows the user to identify the different system elements - especially as they were affected by phase boundaries.

Table 7 shows that:

- The models are better able to provide information needed in early phases of the life cycle than in the later ones.

Most of the models provide the detail needed for system level cost estimates; but none of them rate very highly when the CPCI and CPRC levels must be described. Therefore, as the life cycle progresses and the need for estimates becomes directed toward the components of the system, the model ratings decrease.

The best performer with regard to detail is the Wolverton model. Its matrix of phases and activities executed at the CPCI level is able to provide most of the detail indicated in the WBS. It was not rated as 5 because the model structure does not incorporate a mechanism for accounting for the different WBS levels and their associated overheads. It was also downgraded because it does not identify costs for facilities, training and installation.

TABLE 7 SUMMARY OF MODEL COMPLIANCE WITH AIR FORCE ESTIMATING REQUIREMENTS

ESTIMATING SITUATION	AEROSPACE	BOEING	DOD MICRO	DOTY	FARR & ZAGORSKI	PRICE S	SLIM	TECOLOTE	WOLVERTON
1. CONCEPTUAL									
Scope	2	4	4	1	1	3	5	4	4
Detail	5	5	5	4	4	5	5	5	5
2. CONCEPTUAL									
Scope	2	4	4	1	1	3	5	4	4
Detail	4	4	4	5	5	5	4	4	4
3. CONCEPTUAL									
Scope	3	4	4	2	2	4	5	4	4
Detail	2	4	3	2	2	4	3	3	4
4. VALIDATION									
Scope	3	3	4	2	2	4	5	3	4
Detail	1	3	2	1	1	3	2	1	4
5. FULL SCALE DEVELOPMENT									
Scope	3	3	4	3	3	4	5	3	4
Detail	1	3	2	1	1	3	2	1	4

NOTE: Numbers indicate degree to which the model satisfies the particular estimating requirements. 5 indicates nominal satisfaction of the requirement.

The Wolverton, PRICE S and Boeing models offer more detail than most of the other models. PRICE S allows the separate identification of subsystem costs and includes allocation of the associated integration costs.

The most common failure in the scope dimension was the omission of the Operation and Support Phase. Only the SLIM model includes an estimate of these costs. This is followed in frequency by omission of the Analysis and Design Phases.

As the system develops, the scope of the cost information naturally, becomes less because the initial phases are completed. That is why the models are rated increasingly higher toward the bottom of Table 7.

The best performer in the scope dimension is SLIM followed by the DoD Micro Procedure, PRICE S and Wolverton. These latter three models do not include the O&S Phase in their estimates. PRICE S is downgraded in the first two estimating situations because it does not include the Analysis Phase and part of the Design Phase.

From Table 7 we conclude that:

- None of the models included in this study fully satisfy the Air Force need for information with regard to scope or detail.

A common fault of the models is the failure to properly describe WBS elements that cross phase boundaries. For example, a system design is often changed after the Coding and Checkout Phase begins (Figure 15), but few models or data sets identify the design effort that occurs in the Coding and Checkout Phase. The PRICE S and Wolverton models with their

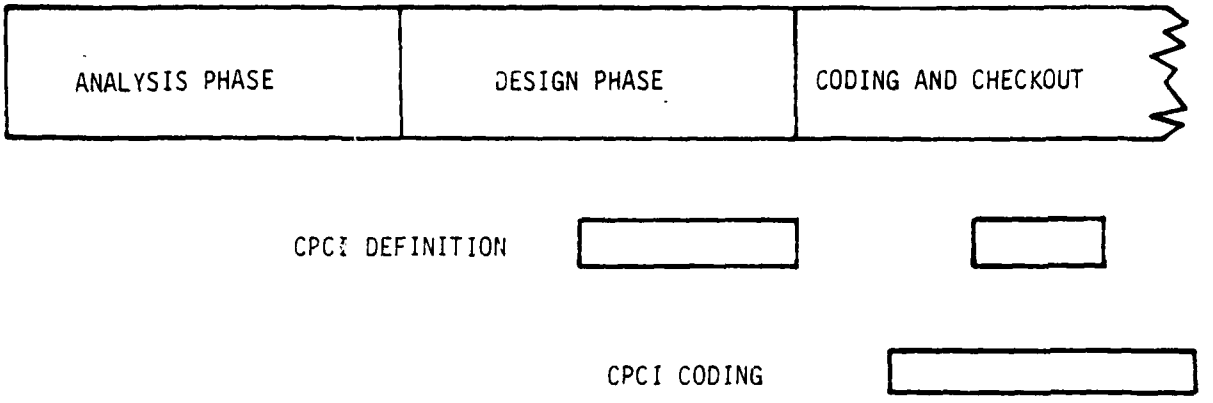


Figure 15 Allocation of Work Breakdown Structure Elements to Life Cycle Phases

matrix representations of the cost elements do permit such distinctions to be made. However, the PRICE S terminology does not allow direct comparisons with the Air Force phases and WBS. In general we can state that:

- The models tend to be phase oriented and do not properly describe activities that cross phase boundaries. This precludes obtaining data compatible with both management planning (phase related) and product cost (WBS).

The tested models, with exception of PRICE S, tend to be vague about how the cost of developing systems relates to the cost of developing system components. As cost analysts well know a system is more than the sum of its parts. Each independently developed component must be designed and redesigned in concert with every other system component. Interface and performance specifications must be analyzed whenever a change is made to any part of the system. This is especially true of large software systems, yet software cost models seldom provide explicit descriptions of these costs.

PRICE S has an operating mode in which individual subsystems can be estimated using the normal techniques and then combined along with a specified integration cost to produce a total system estimate. Both the system and subsystem costs are presented.

The other models simply add subsystems together without regard to size or the number of organizations contributing to the development. It is the user's responsibility to add any integration costs and to properly distribute them.

This leads to our observation:

- Although most of the models use the summation of program, function, or module sizes to make their cost estimates, only one model studied provides for keeping track of the cost on a component basis and accounts for the cost of system integration. None of the models provide for all four levels of system definition called for in the Work Breakdown Structure.

5.2 MODEL ESTIMATING ACCURACY

According to the procedure described in Section 4 and using the Relative Root Mean Square Error measure of estimating performance (Section 3.2), each of the subject models was executed using as many different input data sets as possible given limitations on available time and historical input data.

Appendix C shows the individual results obtained by executing the models. The presentations include estimated and actual values of the outputs for a given project and several performance measures. The following paragraphs present the analysis of the estimating performance. The objective is to identify specific attributes of the model structures or the data sets that are associated with estimating accuracy.

The estimating performance of the subject models for the test data sets is summarized in Table 8. In order to direct attention to structures and to avoid the appearance of making general statements about the prediction performance of the specific models, the models are identified by codes in the charts. This is done to direct attention but there is no intention to hide the results obtained for each model. Therefore, Table C-17, in Appendix C, shows similar information as Table 8 and includes the names of the models.

The analysis of estimating accuracy was accomplished by testing several hypotheses associated with:

- Development environment (Data Set)
- Model structure
- Model calibration
- Use of system size as a model input

TABLE 8 SUMMARY OF MODEL ESTIMATING PERFORMANCE

	RMS ERROR* / MEAN PROJECT SIZE		
	DATA SET		
MODEL TYPE	COMMERCIAL	DSDC	SEL
REGRESSION			
A	1.35	2.11	0.605
B		1.05	
C		16.9	
D		4.92	
E (Recalibrated)	0.643	0.933	0.389
HEURISTIC			
F		0.787	
G		1.26	
H	0.383	1.44	0.297
I		0.927	
PHENOMENOLOGICAL			
J	0.246	0.216	0.865

* RMS ERROR $= \left[\frac{1}{N} \sum_{i=1}^{N} (ACT_i - EST_i)^2 \right]^{1/2}$

Four of the models (A, E, H, J) were executed on all three data sets. The measures of accuracy obtained from these 12 cases were subjected to a two-way analysis of variance [45]. The analyis of variance is a systematic way of inferring the statistical significance of the relative contributions of each model and data set to the total sample error*. The contribution of each model to the total Root Sum Square Error is observed while controlling the contributions made by the different data sets. The contributions of the data sets can be similarly analyzed. This procedure was used to test the first three hypotheses above.

The 12 test cases are tabulated separately in Table 9. Included in the comparisons are two regression models (A is not calibrated, E is calibrated), a heuristic model, H, and the phenomenological model, J. The heuristic and phenomenological models have calibration modes that can be used before making estimates.

The two-way analysis of variance produces an inter-column F statistic of 1.97 and an inter-row F statistic of 1.95. The null hypotheses that the row and column effects on the total mean square error are zero, can both be rejected with an 80 percent level of confidence. This means that taken as groups there are differences in estimating accuracy among the models and among the environments represented by the three test data sets.

---

* Citing any statistical procedure implies certain assumptions about the characteristics of the populations represented by the test subjects. The analysis of variance is restricted to normal populations with equal variances. There is an assumption of linearity of the contributions of group differences to the total sample difference, independence and others. We cannot be certain that these conditions are even partially satisfied. Therefore, the presentation of the statistical results is not made as proof, but only as an indication of possible support. Of course, even under rigorous satisfaction of all conditions, causality is never proved. Other observations will be offered in this section without statistical justification, but only justified by conforming to intuitively acceptable patterns.

TABLE 9 EFFECTS OF ENVIRONMENT AND MODEL TYPE ON ESTIMATING PERFORMANCE

RELATIVE RMS ERROR

	DATA SET			
MODEL	COMMERCIAL	DSDC	SEL	MEAN
A	1.35	2.11	0.605	1.36
E	0.643	0.933	0.309	0.628
H	0.383	1.44	0.297	0.707
J	0.246	0.216	0.865	0.442
MEAN	0.656	1.17	0.519	

Comparisons among the individual row and column members were made to learn how the different models and data sets contributed to the overall results.

Table 10 is a presentation of the error totals by rows and columns. The table entries are the differences between the marginal values. For example, the first row and first column entry in the model table, 2.13, is the difference between the total error on all three data sets for Model A (4.01) and the similar measure for Model E (1.88). The null hypothesis that the difference is zero can be rejected with an 80 percent level of confidence for values in the model table greater than 1.73 and in the data set table greater than 2.00. Asterisks indicate the significant values.

Table 10 indicates that significant differences in estimating accuracy exist between Model A and each of the other models. However, the accuracy differences among the other models are not significant. Model A (Aerospace) is not recalibrated to any of the test data sets; and Model E has the same form as A ($MM = aI^b$) but has been recalibrated. Therefore, the results indicate the effect that recalibration has on estimating performance.

The recalibration of the form used by Model A produces a model that has the same estimating performance as the other two models. This suggests that:

- Recalibration is the primary factor contributing to the difference in estimating performance experienced by the models in Table 9 and
- The contribution to estimating accuracy related to model structure is not significant when each of the model structures have been calibrated to a given data set.

TABLE 10

PAIRWISE COMPARISONS OF ESTIMATING PERFORMANCE

Table Entries are Differences in total RMS Error

COMPARISONS OF MODELS

Model	Total RMS Error	A 4.01	E 1.88	H 2.12	J 1.33
A	4.01				
E	1.88	2.13*			
H	2.12	1.89*	0.24		
J	1.33	2.68*	0.55	0.79	

COMPARISONS OF DATA SETS

Data Set	Total RMS Error	Comm. 2.62	DSDC 4.70	SEL 2.08
Comm.	2.62			
DSDC	4.70	2.08*		
SEL	2.08	0.54	2.62*	

* Difference is significant with at least an 80 percent level of confidence.

The second statement derives from the fact that the three calibrated model structures are very different. E is the simple form $MM = aI^b$, H is the PRICE S model and J is SLIM.

The pairwise comparisons of the estimating performance of the models among the different data sets indicates that the accuracy measurements are significantly different for two data sets. This indicates that:

- The development environment is a significant factor affecting the relative estimating performances of the models tested.

A given model structure will perform better in some environments than others. This finding indicates the necessity of learning the specific attributes of a development environment that determine when one or another model structure should be used. This view is further supported by the performance of Model J, SLIM, shown in Table 9. Model J is the best performer for the first two data sets and the worst performer on the third. It is essential that we identify the characteristics of the model structures relative to those of the development environment that affect the ability of the model to make accurate predictions. This result also substantiates the need for making accuracy evaluations on as broad a range of environments as possible. In effect we are saying:

- The effect of development environment on estimating performance precludes the possibility of obtaining generally applicable measures of the performance of any model or model structure without applying additional controls.

The measurement of the effect of recalibration on estimating performance was repeated using some of the other results in Table 8. Since some of the models are recalibrated on each data set and others are not, the estimating accuracy of one set, the calibrated models, was compared with the non-calibrated ones. In order to make the comparisons on the same basis only one data set was used and the models were separated by type.

The four non-calibrated Regression type models (A, B, C, D) with a relative RMSE of 6.10 were found to be not significantly different from the 0.933 value obtained for the calibrated Regression models (E). This is because the large range in the error values has a correspondingly large variance which reduces the statistical significance of differences between the mean and any given value. This is true even when Model C is eliminated from the group.

The exercise was repeated for the Heuristic model types. In this case the calibrated models include F, G, and I. These with their relative RMSE of 0.991 were compared with the 1.44 value obtained with Model H. The difference is not significant.

These two experiments conducted on a single data set failed to substantiate the findings previously obtained by executing four models using all three data sets. However, there is an important difference in the two investigations. In the first analysis the effect of calibration was obtained by using the same model structure to represent both non-calibrated and calibrated models. In the second analysis a group of structures was used to represent the non-calibrated models. The results may only indicate that there is too much variation among the models to allow a significant comparison of estimating performance. This also indicates that better criteria for stratifying the model structures are needed. This view is also supported by the fact that the first analysis was more limited in the number of model structures it included. Considering the effect of calibration on estimating performance it is necessary to know the portion of the variation in performance among the model structures that would be eliminated by calibration. Only after the model structures are recalibrated would it be possible to explore the effects on accuracy of such model attributes as: number and types of input variables, level of the initial estimate, the method of making the initial estimate and the method for making subsequent estimates.

Examination of the results in Table 8 suggests some additional findings that are presented without statistical justification. Considering the previously observed effects on performance related to recalibration and environment, these inferences must be considered tentative.

The only model in the test group that does not use a form of code size as an input (G, DoD Micro Procedure) has estimating performance that is comparable to the other models of its type. This suggests that:

- Models that do not use size as an input may perform as well as those that do.

If this is true there may be an increase in accuracy obtained by using a non-size input in the early phases of the life cycle when size is known less precisely (see Section 7.1).

TABLE 11

AVERAGE ESTIMATING PERFORMANCE

	Average Relative Root Mean Square Error
All Models and Data Sets	1.930
Non-Calibrated Models	3.260
Calibrated Models	0.592

Table 11 indicates that:

- The average RMS Error for all tested models is unacceptably large for Air Force estimating purposes,
- The use of models that are not calibrated to a given development environment can lead to very large estimating errors,
- The best performance obtained by any group of the models tested (calibrated, RRMSE=0.592) is not adequate for Air Force needs.

When using the Relative RMS Error to describe estimating accuracy, it should be understood that the values do not represent expected values of error for estimating situations. The RMS Error is a weighted measure that penalizes large deviations from predicted values. Appendix 3 shows that large deviations are common among the models and justifies the use of this statistic.

6 ANALYSIS OF RESULTS

6.1 ENVIRONMENT

A model is an abstraction of some real process that we try to represent using some selected variables and a hypothesized relationship among them. When we construct the model we hope that we have found the important variables. That is, that the forces at play that determine the outcome of the process are adequately represented by the model. We never know for sure if this is true; that is, if the model is valid for all circumstances. We can only state that if it is observed for a long time over a large part of the input domain and if its behavior is consistent with experience, we begin to feel comfortable that it will always behave as the prototype behaves. Then, the model is believed to be a valid representation of the system behavior.

However, we never know when we may wander away from the domain on which the model is valid. The Ptolemaic model of the solar system was believed valid until deviations between the model and observation were established. This process took hundreds of years.

The Newtonian model of dynamic behavior was believed valid until the relativistic domain was encountered. In both cases behavior was in concert with theory only until phenomena which had always been present became evident. Then the models were understood to be either special cases of a more general representation or simply invalid.

Models of software development are in a primitive state. It is not possible to completely explain observed phenomena in controlled environments let alone make general statements. Therefore, we must view these findings as indications of possible relationships and nothing more.

The forces acting on a software development project are many, complex, often subtle and even counter-intuitive [8][46]. They reflect human performance and its variability among individuals. Individual

performance itself varies according to many circumstances. The forces also reflect group behavior and the availability of resources. The total of all factors affecting performance constitute the development environment. If the personnel, equipment and management structure in a given organization remain relatively unchanged over a period of time, we have a better probability of obtaining valid models of the organizational behavior. This is because many of the forces that may have profound effects on performance are not changing and therefore need not be considered explicitly. Their effects will be included in the parameters of the model. However, if the organizational environment is changed, the model may become invalid and its behavior may depart radically from its prototype.

The point of this discussion is that the results obtained here were observed for three different organizations that constitute three different development environments. We have observed that the collective behavior of the model types differ among the environments. This suggests we should be very cautious about drawing inferences about the behavior of any model in the evaluation and apply them to another environment such as our own organization. Notice that the rankings of the models on the basis of accuracy was not consistent for the three data sets. Some of the models moved from best to worst depending on the test data. We have observed some behavior but we must be very cautious about making any generalizations. We have the basis for making some hypotheses, but we haven't proved anything. What has been demonstrated in this study may be the result of the choice of environments and project types.

6.2 THE EFFECTS OF INPUT ESTIMATING ERRORS

Section 3, Evaluation Procedure, indicates that missing input data were estimated when necessary to execute the models. It is desirable to test the effect that such estimates may have on the measurement of comparative estimating accuracy.

The model estimates are subject to random errors associated with the values of the inputs. This means that the observed differences in estimating performance are expected values and they are significant subject to some uncertainty. It is possible that in some cases the uncertainty may be great enough to prevent acceptance of the result. It was not possible to establish measures of variation for all the model estimates for all the data sets. This should be done. It was possible to do it for two models and the effect on the comparative accuracy measurement is presented here. However, the presentation is more an illustration of how the effects of errors in the input values should be included in the analysis of the model estimating accuracy rather than a representative finding.

Two models, F and G, were selected to test sensitivity to input errors. These two models use different inputs; and each has several estimated inputs. As shown in Table 8, Summary of Model Estimating Performance, models F and G have relative RMS errors of 0.787 and 1.26 respectively. We want to determine whether the variations in the predictions produced by the uncertainties in the input values allows us to accept the difference with reasonable confidence.

Tables 12 and 13, taken from Appendix D, present the inputs for the two models. Model F has three estimated inputs and Model G has five. The tables contain the estimated error associated with each input.

Each model was executed 100 times using input values selected at random from normal distributions defined by the given means and standard deviations. The distributions of the sampled values were truncated to admit only positive values.

The executions produced the following results:

TABLE 12. INPUTS FOR MODEL F, BOEING COMPUTER SERVICES

SYSTEM	REPT $\hat{E}$	STMTS $\hat{S}$	LOGIC $\hat{E}$	STMTS $\hat{S}$	PROG. $\hat{E}$	EXP. $\hat{S}$	NO. PROGS.	EXIST. SW	F.O.	HOL	MACRO-CODE	MACRO-FORMS	ON-LINE DEBUG	NO DEBUG TOOLS	ON-LINE CODE/DATA ENTRY
1. DC	1751		382		5.0	3.7	1	N	N	Y	N	N	N	N	N
2. DK	6071		1126		5.0	3.7	2								
3. DS	3500		305		5.0	3.7	1								
4. DU	2773		348		5.0	3.7	1								
5. FB	14421		2394		6.9		2								
6. FD	2814		547		8.6		3								
7. FE	6512		1827		9.9		1								
8. FF	3887		444		9.6		1								
9. BH	8893	497.5	1477	281.5	1.6		1								
10. BB	3665	485.3	594	274.7	2.8		1								
11. GG	6057		1056		5.0	3.7	2								
12. BI	1526	491.2	232	278.0	5.0	3.7	1								
13. ZP	1783	490.1	276	277.4	2.6		1								
14. US	19604	621.9	3288	352.0	0.7		4								
15. JD	4703		485		5.0	3.7	2								
16. QD	11279		2151		1.9		1								
17. DJ	3568	485.4	577	274.7	5.0	3.7	5								

TABLE 13. INPUTS FOR MODEL G, MICRO ESTIMATING PROCEDURE

SYSTEM	NO. I/O FILES		COMPLEXITY		EXP. AVAIL		JOB KNOW.R		JOB KNW AV.		TURN AROUND		SYS. FCTR	
	$\hat{E}$	$\hat{S}$	$\hat{E}$	$\hat{S}$	$\hat{E}$	$\hat{S}$	$\hat{E}$	$\hat{S}$	$\hat{E}$	$\hat{S}$	$\hat{E}$	$\hat{S}$	$\hat{E}$	$\hat{S}$
1. DC	59		144.5		1.08	0.500	0.5		0.72	0.363	0.8		3	
2. DK	308		209.5		1.08	0.500	1.0		0.72	0.363	0.8		3	
3. DS	158		168		1.08	0.500	0.5		0.72	0.363	0.8		3	
4. DU	151		129		1.08	0.500	1.0		0.72	0.363	0.8		3	
5. FB	709		536.5		0.75		1.0		0.5		0.8		3	
6. FD	220		138.5		0.75		0.5		0.5		0.8		5	
7. FE	185		289		0.75		1.0		0.5		0.8		3	
8. FF	143		168.5		0.75		1.0		0.5		0.8		3	
9. BH	386	48.4	390	70.3	1.75		1.0		1.0		0.8		3	
10. BB	161	47.2	177	68.6	0.75		1.5		0.5		0.8		3	
11. GG	280		325		1.08	0.500	0.89	0.350	0.72	0.363	0.8		3	
12. BI	69	48.0	90	69.7	1.08	0.500	0.89	0.350	0.72	0.363	0.8		3	
13. ZP	80	47.7	100	69.2	1.75		1.0		0.5		0.8		3	
14. US	847	60.5	826	87.9	1.75		0.5		1.5		0.8		3	
15. JD	262		178		1.08	0.500	0.5		0.72	0.363	0.8		3	
16. QD	480		617.5		0.75		1.5		1.0		0.8		3	
17. DJ	157	47.2	173	68.6	1.08	0.500	0.89	0.350	0.72	0.363	0.8		3	

Relative RMS Error	MODEL F	G
Mean	0.787	1.26
Standard Deviation	0.00893	0.0458
Stand. Dev. of Mean	0.00282	0.0102

The hypothesis that the actual difference between the two means is zero can be rejected with a confidence level greater than .999. This finding indicates that the relative estimating performance between the two selected models is not a random result arising from uncertainties in the model inputs.

6.3 MODEL CALIBRATION

We have shown that the calibration of model parameters may be as important as model structure in explaining estimating accuracy. Two of the models tested incorporate calibration into the estimating process. That is, these models require or suggest that data representative of the development environment be used to compute model parameters before using the model to estimate new projects. In effect the creators of these models are saying that in addition to certain fixed parameters that are permanent parts of the model structure, there are additional values that describe the effects of the circumstances under which a project is executed. These values are ideally constant in a stable environment, but some users apply judgement based on experience to the selection of values for a given estimating situation.

Model H, PRICE S, utilizes two calibration parameters, one sensitive to to total cost, the other to development time [47]. The user of the model is instructed to execute the calibration mode of the model to obtain values of the two parameters from past projects that are representative of the expected development environment.

Model J, SLIM, has one calibration parameter which is said to represent the efficienty with which a given organization can produce a given type of system [48]. This model also has a calibration mode for obtaining the parameter from project experience.

The calibration of each of the models did not produce constant values of the parameters. It could be argued that the reasons the parameters varied was that the environments were not similar. But this is circular reasoning. Since in both instances there is no way to measure the calibration parameters directly, we can only observe their effects on the model estimates. If deviations in the calibration parameters are proof that development environments are different from one another when it was assumed they were representative of the development environment, then we must admit that we don't know what constitutes the a priori indicators of environment. If that is t,ue, we have no way to know if any new project will be represented by one past environment or another and will not know which parameters to use for estimating.

It is necessary in any calibration of model parameters to know when calibration is necessary and when any given parameters are applicable in an estimating stiuation. In the case of models H and J the calibration constants vary significantly among projects within a given organization. This coupled with the sensitivity of the model outputs to changes in the calibration constants makes the solution of the calibration problem the key to successful model performance.

Calibration of models H and J was initially accomplished by randomly selecting several projects from a set of test projects and executing the calibration modes of the models. Significant variation in values of the calibration parameters were obtained for both models. Since the number of projects available fo measuring estimating accuracy were limited, it was decided to systematically vary the calibration variables

to obtain the best estimating performance. This biases the evaluation and produces the best possible performance for a given data set, but we did not want to compromise either model's performance because of a chance selection of projects for calibration. Observing the effects of changes in the calibration parameters on the model predictions simulates the learning process that would occur as experience is obtained using a model in the same environment. In the case of each model the experience mode gave better estimates than the model calibration mode.

The recalibration, E, of the Aerospace model (A) was accomplished by a linear least squares fit of the logarithm of the program size in lines of source code for the three data sets. Since the parameters obtained were used to show estimating performance on the same data sets, the recalibration results are comparable to those obtained using the extended calibration procedure on models H and J.

When model structures are calibrated to a given development environment, the effect of structural differences tend to disappear. This is true at least for an accuracy measurement derived from a total effort estimate. However, this ability to predict is very sensitive to the environment and it is not known how the success of a model in one environment is related to success in another. We don't know how to measure an environment's attributes to know if it belongs to the same population represented in a given data set that was used to obtain the model parameters. We only know that for the case where it is known that a project is a member of a data set, the performance is within observed limits. But even within a calibration data set the differences in estimating accuracy can be large. We have shown that calibration is effective in increasing estimating accuracy, it we don't know when calibration is necessary or which historical projects to include in a calibration to obtain the best accuracy. We have seen some classifications of development projects according to such descriptions as commercial, scientific, and time critical [22] [23] [49], but these definitions are never explained.

The problem of selecting a model for a given estimating situation may be stated more clearly in terms of specific models. For example, assume we must select among models E, H, and J to make an estimate of software cost. This presumes that data are available to calibrate the models. First we must decide if the development environment is comparable to one described by one of the three test data sets. If the development environment is similar to SEL, Models E or H should provide the most accurate estimate. If the environment is more like the Commercial or DSDC environments, Model J should be the most accurate. The problem is that we don't know which of the many attributes that may describe a development environment are sufficient to determine the equivalence of two environments for estimating purposes.

6.4 THE USE OF UNMEASURABLE VARIABLES AND PARAMETERS

It is possible to identify two points of view that are evident in the choices of model predictor variables and parameters. One group of models includes only variables that can be readily measured at some time during or after the development of a software system. Definitions may differ in detail, (e.g. program size) but they can be measured. The important consideration is that the estimate of the variable can be verified subsequently and if the prediction is off, we will know whether it is because the estimate or the model was wrong.

The second type of model includes subjective variables that may be representive of important attributes of software development but are not expressable as measurable quantities. An example would be a variable representing the difficulty or complexity of the development effort relative to the ability of the development group. It may be possible to obtain a consensus among analysts about what these values should be, but the values are never observable before or after the software development. Therefore, if the

estimate subsequently differs from experience, we cannot know if the difference was because the predictor was not estimated properly or the model failed to perform. We can discuss whether the initial extimates were appropriate given past experience and the characteristics of the software and the development environment, but the results are always subject to interpretation. This is especially true if the model estimates are sensitive to the subjective parameters.

The exclusive use of measurable variables may unnecessarily limit the development of model structures. It may be that it is not possible, given the state of the software engineering art, to identify and quantify the variables that determine the cost of developing software. On the otner hand, too much reliance on subjective inputs may perpetuate the concept of estimating as a black art that cannot be explored objectively. It may be best at this time to pursue a policy that favors the maximum use of measurable predictors but recognizes the possibility of using subjective inputs. Subjective estimators supported by carefully documented guidelines for their valuatation may provide reliable estimates. Therefore, we should accept such models as legitimate interim steps which may provide accurate estimates and possibly insights into more objective measures.

The problem remains, however, of making an objective measurement of the prediction accuracy of models that include unmeasurable inputs. There is a tendency to play with the subjective models until good results are obtained. The fact that good results can be obtained may be significant by itself, but model sensitivity may ensure this. The process may be compared to the natural use of the model in any stable prediction environment. But the fact remains that the comparisons between the models are not being made on the same basis. Our conclusion is that the only evaluation of a subjective model (one that includes one or more major parameters that can't be measured) is to observe, if in a stable environment where the type of software being developed is the same and there are no upheavals in the organization or the personnel, whether the parameters benave in a predictable fashion. If this is true, it would seem that the

model can be used in that environment for the tested type of software. But if given a stable environment, the parameters that give the best a posteori predictions do not behave in a moderate, predictable fashion, then the model probably is not a useful tool for that environment.

In either case no general conclusions may be drawn about the values of the models in other environments or for that matter in the same environment if the type of development changes significantly. Here significant is not definable because there are no reliable principles that define the domain of the predictors. Therefore, we never know when they may become unreliable or invalid. This will be true to some extent for the objective models, but it is of much greater import for the subjective models.

Our evaluation objective is to learn if certain model structures can be demonstrated to be better predictors than others and if so to recommend how future model development should proceed. Our interest in the subjective models should they exhibit positive qualities is perhaps to learn if the subjective variables can be quantified or at least to learn the boundaries of the regions within which they may be used successfully.

6.5 APPLICABILITY OF THE EVALUATION

The evaluation is made using data obtained from three different environments. They were obtained using questionnaires to supplement other measures (e.g. time reports). The system developers were contacted whenever data seemed inconsistent with itself or failed to satisfy the analyst's intuition. However, as was the case with the SDC data and as Nelson [50] expressed so well:

> "All the data used from both the statistical analysis and the literature were data of opportunity, i.e., we took what we were able to get in the time available. Hard data on the costs of computer programming . . . are scarce commodities both in computer programming

> organizations and in the published literature. Few numerical data are recorded; fewer yet are recorded under 'controlled' conditions, and still fewer are suitable for generalization to other situations . . . The respondents to the questionnaire were under no obligation to assure completeness and accuracy even when data were readily available. Because they were suspect, some of the data collected were rejected prior to the analysis. But even those data used in the analysis are likely to have a variation in reliability . . ."

This is a frank and somewhat negative evaluation of the data quality, but before anyone hastens to disparage the results because they don't support their own experience or because they fail to show their favorite model to be as good a performer as they believe it to be, we should state that the data used were of as good quality as any other available for cost analysis and considerably better than most. Given the opportunity to analyze most software development data in detail [51] [52] we would find the same kinds of deficiencies that we have expressed in assessing the data on which this analysis is based. If someone has executed a model that has historically performed better for them than is presented here, then we would argue that there is a fortuitous fit between the characteristics of the model and the environment in which it is being used. Nothing in this report should be construed as a general statement about model performance. We can only describe the models' performances as they were represented in a very careful analysis that used good quality data and an objective comparison.

This evaluation has shown what other researchers have already expressed [31] [43] [47] [48] [49] [50] [53] [54] [55]: that model performance given the limitations on standard development procedures, definitions and understanding of the cost driving factors is very much environment dependent.

We think the results give a good indication of what can be expected when these models would be used by an outside agancy such as the Air Force when trying to make estimates in support of the Major Weapon System Acquisition Process. This was our objective and we think we accomplished it. We readily concede that given the opportunity to calibrate a given model over a period of time in a stable environment, that better performance might be obtained than is presented here, but that is a different estimating situation from that considered in this analysis.

7 RECOMMENDATIONS

Four of the five cost estimating situations described in Section 2 occur when very little is known about the development environment. Given the significance of the effects of environment and calibration, it is necessary to develop methods whereby the Air Force can overcome the disadvantage of operating at a distance. The Air Force must identify and obtain the data items that will ensure accurate estimates in any specific environment. These would include data that characterize the past performance in that environment as well as items of importance to the project being estimated. The Air Force must obtain the visibility into a cost estimating situation that is presently available only to the persons who are members of the organization and who have first hand experience with that organization's performance.

Under this approach model structures will be sought that are easily calibrated to a given organization using auditable historical data provided by the organization. These data and others specific to a given project would enable the Air Force to validate a proposal for software development. A collection of such data would characterize any group of organizations and would be used in the initial phases of the life cycle.

The following recommendations describe a course of action that will provide the above capability. The objective of the recommendations is to place the analysis and synthesis of Air Force software cost estimating models on a systematic basis.

7.1 MODEL DEVELOPMENT

The results of the accuracy evaluation suggest that the best way to make cost estimates is to use the simplest model structure and to calibrate its parameters to represent the development environment. However, this approach fails to consider several factors:

- The measurement of estimating accuracy in the present study does not consider the need to estimate the elements of the Work Breakdown Structure,

- None of the model structures achieved the needed level of accuracy,
- The accuracy of the models in this analysis is overstated because it reflects no error attributable to uncertainty in the input values. This error changes during the life cycle and is different to each input.

It is very likely that the incorporation of these considerations into a further analysis would affect the findings regarding the model structure. Whereas a simple structure may be adequate for estimating total development effort, a more complex structure is needed to define cost elements for a single phase at a lower level of the WBS. Therefore, the present findings must be considered inconclusive regarding the effects of particular structures on prediction accuracy. The effects of calibration and environment are of a comparable magnitude to the variations among the individual models within a type category. Additional studies are needed to quantify the following effects on prediction performance:

- The level of the initial estimate.
- The method of making the initial estimate.
- The method of making subsequent estimates.
- Alternatives to the size measure.

One analysis that is suggested by the present study is the determination of the influence on estimating accuracy of the use of size of code as an input. Models that use size as an input should be further classified according to whether the code is decomposed into types (e.g. as in Boeing, Wolverton, PRICE S). The basis for comparison in each case should be the initial estimate because in most models a process different from the initial one is used to manipulate this value to obtain the full scope of the model outputs. It is necessary to evaluate the two methods independently.

The DOD Micro Procedure, Farr and Zagorski and Wolverton models should be calibrated to the evaluation test data. This would provide seven calibrated models. They should be classified as follows:

- Size of code
  - Types of code
  - No types of code
- No size of code

The results should establish to a higher degree of confidence than was possible in the present study the difference in estimating accuracy, if any, afforded by the use of code size as an input. They would also indicate if defining the type of code to be written increases estimating accuracy. Other experiments should be designed to test the performance of different methods of proceeding to other cost estimates.

The results of these investigations should be used to establish the basic attributes of model structures that have demonstrated high accuracy. When this has been accomplished using existing models, it will be possible to design a series of second generation models based on the structures that have performed well in systematic tests. These models will be coordinated with the development of better data sets which will permit more complete exploration of structures and more comprehensive testing.

7.2 DATA DEFINITION AND COLLECTION

Data availability and quality has been a major limiting factor in cost model development. This evaluation has indicated the importance of data definitions to the interpretation of model performance. The recommended direction for future model development puts additional requirements on data. The Air Force has recognized the need for software data and has taken a major step in the establishment of the data repository at the Data and Analysis Center for Software (DACS). We recommend that data collection efforts continue to be focussed at DACS and that the model

development activities be used as the basis for establishing data reporting requirements under software development contracts. The DACS is the ideal catalyst for coordinating the dissemination of information describing progress toward both objectives.

If program size is determined to be an important factor in making accurate estimates, DACS should be responsible for maintaining standard definitions under which project data could be reported and tabulated. The same should be done for any other inputs or outputs required for cost estimating. The present role is passive. Given a productive research program the definition of data elements should become active. The Air Force should take charge of defining its needs for estimating data.

Software data reporting should become an integral part of the contracting process much as operating costs are now. Items and formats should be defined by the Air Force and provided routinely by the contractors. Audits should be possible if necessary to substantiate the reported values. A well-designed data reporting scheme (e.g. [43]) should not be burdensome on the contractors and should pay for itself in better planned and managed system development projects.

A recommended data collection project is the acquisition of estimates of input variables during the life cycle. The results in the present evaluation were obtained using actual values usually recorded after a project was completed. This reflects minimum uncertainty in the inputs. Therefore, the accuracy of the different models is higher than it would be if we included the precision with which the inputs are known. The different inputs are not known equally well at the different times that estimates are made. This biases the results in favor of those models which use inputs that are known accurately only late in the development cycle.

APPENDIX A

DETAILED MODEL DESCRIPTIONS

AEROSPACE MODEL

Description of the Model

The model was developed using regression techniques applied to data from software development projects characterized by one-of-a-kind computers, limited support software, special languages and severe memory size and speed requirements. The data were stratified into two groups. One group contained 13 projects for the development of real time software identified as primarily large-scale airborne and space applications. The second group consisted of 7 operational support programs presumably without the size and speed requirements of the first group.

The model description is not clear concerning the exact composition of the estimate of effort required to develop the software. Only the total effort is estimated. The estimate is made using a relationship of the form:

$$MM = a\ (\text{Instruction})^{b}$$

where the constants, a and b, are determined by regression analysis.

The estimating relationships are:

Real Time Software

$$MM = 0.057\ (I)^{0.94}$$

Support Software

$$MM = 2.012\ (I)^{0.404}$$

where:

MM = total development effort, manmonths

I = number of instructions (independent of language).

Reference

T. G. James, Jr., "Software Cost Estimating Methodology," Proceedings IEEE VAECON '77, May 77, PP 26-27.

Boeing Computer Services

Description of the Model

The Boeing Computer Services (BCS) software cost model estimates total project effort from a table of productivity rates that associates different types of software. The rates are applied to the sizes of the delivered programs to obtain estimates of the direct effort required to develop the programs.

The BCS model works best for aerospace types of systems. The most reliable estimates of the inputs are obtained from project planners who have related experience and a good knowledge of the system requirements. The method begins with decomposition of the systems into functions and modules. This requires knowledge and a certain amount of component definition, i.e., how the system functions will be performed and how they should best be divided up for development.

The project leader is asked to divide the system according to its composition among the following types of software:

- Mathematical Operations
- Report Generation
- Logic Operations
- Signal Processing or Data Reduction
- Real Time or Executive (also Avionics Interfacing)

These assignments are based on experienced judgment and are subjective.

Then an estimate is made of the total number of delivered instructions. BCS experience indicates that come care must be taken with this estimate because project leaders tend to think in terms of developed code and not delivered code. But they have found it necessary to make this distinction because non-delivered code such as test drivers is normally not tested and documented as thoroughly as delivered code and consequently requires substantially different development resources.

The method then applies different productivity rates for each type of software to obtain development effort. The rates were obtained empirically from over two dozen sources. Some of these, for example, include Wolverton, SDC, TRW, and Boeing projects such as Lunar Orbiter. Code types were identified and productivity was obtained from total project effort data. The factors were obtained by curve fitting. The results were confirmed by knowledgeable people. The productivity rates range from 6 to 40 manmonths per 1000 source statements.

The total development effort is divided into project tasks according to a fixed schedule.

	% OF TOTAL
Requirements Definition	5
Design and Specification	25
Code Preparation	10
Code Checkout	25
Integration and Test	25
System Test	10

The resulting values are then adjusted for 9 conditions:

Reimplementation of existing software
Follow-on contract
Number of programmers
Higher order language
Macro language
On line code/data entry
On line debugging
Poor or no debug tools
Programming experience

The adjustment factors for each condition are applied to the appropriate task efforts.

The total development effort includes manmonths for all direct personnel (except final documentation) and first line supervision (however, it should be noted that many of the sources of productivity data were non-Boeing environments). A typical group would have a supervisor and 6 to 10 programmers. Larger projects with higher levels of management would not include these managers in the total effort prediction.

Outputs

Raw Development Effort. Man-months to fully check out, test and document software of a given type and number of statements. The total development effort is the sum of the development efforts required for each type of code. This is a raw value that is derived from estimates (by type of software) of numbers of delivered statements of new code divided by productivity rates. The rates are different for each type of software. The development effort is allocated among six development phases. The man-months in each phase are subsequently adjusted to account for existing software, higher, order language, programming experience and other factors.

Distributed Raw Development Effort. The apportionment of the Raw Development man-months according to fixed percentages.

Task or Development Phase	Percent of Total Effort
Requirements Definition	5
Design and Specification	25
Code Preparation	10
Code Checkout	25
Integration and Test	25
System Test	10

Adjusted Development Effort. The man-months of development effort for each software type separated into development tasks or phases and adjusted for nine product and environmental factors.

1. Reimplementation of existing software
2. Follow-on contract with current customer
3. Number of programmers
4. Higher order language
5. Macro language
6. On-Line code, data entry
7. On-Line debugging
8. Poor debugging tools
9. Programming experience.

Computer Time. Two time estimates are provided:

- Stand-alone time (dedicated computer)
- Computer resource units (not defined)

Stand-alone or dedicated computer availability may occur in developments utilizing minicomputers or special purpose computers that are totally devoted to a given application. The cost of the computer or the access to computer facilities is not shared with any other software development project.

Computer resource units are used to measure the portion of a large multi-user computer facility appropriated for a single task. Usually, the computer operating system measures the amount of time that the task uses computer memory, different peripheral devices, software packages, and other resources such as multipart paper, tapes, etc. and uses an algorithm to charge the user according to the portion of the total system that was made available. The definition of a computer resource unit depends on the computer equipment configuration, the relative costs of the devices, and the method of allocating fixed costs. CRUs are not in general comparable from one computer installation to another.

Inputs

Product Related Inputs

Number of Statements. The number of statements to be written and delivered to the customer. The count includes only new code and excludes, "test drivers, test data bases, translators, simulators, etc." that are written but not delivered. These are separated from the delivered code because they do not undergo the same level of testing and documentation. They are accounted for in the estimating procedure by a subsequent adjustment to the development effort.

The number of statements describes executable statements, but includes storage defining statements (e.g., FORTRAN COMMON). An allowance should be made if the specifications describe unusually severe requirements for commentary within the code.

The number of statements are counted according to five types of software:

- Mathematical Operations
- Report Generation
- Logic Operations
- Signal Processor or Data Reduction
- Real Time, Executive, or Avionics Interfacing

Resource Related Inputs. The following outputs are used to modify the estimates of the development effort required for each type of software They are used to identify constant adjustment factors that are associated with the different development phases. Two of the inputs, Number of Programmers and Programming Experience, are cardinal numbers; the others are either applicable to the planned development or not.

Number of Programmers. Three sizes of development team are described:

- 1-2
- 6-10
- More than 20.

Each team size is associated with a factor that increases effort with increasing team size.

Programming Experience. The relative experience of the group in the technical descipline being programmed is identified by one of three levels:

- Entry-Level
- Moderate
- High

The following inputs describe factors or conditions that may or may not be appropriate to the development.

- Reimplementation of existing software
- Follow-On contract with current customer
- Higher-Order language (seasoned compiler)
- Macro-Language
  - In coding
  - Forms for document
- On-Line debugging
- Poor (or no) debugging tools except dumps

Reference

R. N. E. Black, R. P. Curnow, R. Katz, M.D. Gray, BCS Software Production Data, NTIS, AD A039852, Mar 1977.

DOD MICRO ESTIMATING PROCEDURE

Description of the Model

The primary estimating relationship comprising the DoD Micro Procedure can be described as the ratio of a factor representing the software to be developed or changed and a productivity measure.

The model form suggests that effort increases directly with the number of input and output configurations operating on the system being built. Effort also increases with the number of routines being created or modified weighted by their difficulty. The total effort is scaled according to the amount of work that must be done in entirety as opposed to modification of an existing system.

The number of days needed to deliver the product (effectively the days of effort per unit of product) depends on the general experience and accomplishment of the development group (measured by their job classifications) weighted by their knowledge of the problem to be solved relative to the knowledge required. One other factor that directly affects the productivity is the ease of access to the computer (measured by turnaround time).

The basic form of the estimating relation for software development time is:

Net Development Time = (Product) ÷ (Productivity)

Where: Product is a measure describing the effort to be performed.

Productivity is the rate of creating the product from the application of personnel time.

Product = (Number of Formats + Weighted Number of Functions)
x (Effort Relative to a New Development)

The terms in parentheses along with the following terms are defined in the discussion of model inputs below:

$$(\text{Productivity})^{-1} = (\text{Work Days per Unit of Product for a Staff with Average Experience}) \times (\text{Job Knowledge Required}) \times (\text{Job Knowledge Available}) \times (\text{Access})$$

The result is the total hours required for code development. Presumably this means detailed design, coding, and unit testing.

$$\text{Gross Development Time} = (\text{Net Development Time}) \times (\text{Other System Factor}) \times (\text{Non-Project Factor} + \text{Lost Time Factor})$$

A value of 1.8 is recommended for the other system factor. This factor represents the effort needed to convert the code development time to total development time. This value is representative of an observed range from 1.2 to 2.1. Total development includes analysis, design, coding, testing and documentation. It is the sum of the project direct charges. Whether this includes support hours for clerical and other functions is not clear, but any given organization could include these by modifying the 1.8 factor.

The net development time accounts for the time lost from normal scheduled working hours for leave, sickness, holidays, and non-project assignments. These add 25 percent to the total development time. There is also a 10 percent efficiency factor (coffee breaks, time cards, code rework, etc.). The code rework should probably be handled elsewhere. It is probably included where it is to make the 10 percent palatable. It should be included in the gross size adjustment and the 1.8 factor.

The effect of these adjustments is to estimate the number of personnel who must be assigned to the project to ensure delivery of the total development hours. These factors are organizational specific.

Although the resource estimating procedure includes weighting factors for the input and output formats by type of device (see subsequent discussion), the factors have a value of one in each case. Therefore, the model describes a linear relationship between the total number of file formats and the effort required to implement them. It may be that future versions of the model will weight the types of file devices differently. Then the effort required to implement a report format may be different from the effort required for a card format.

Program complexity, which is the second term in the product measure, is the weighted sum of the functions to be implemented. The weights depend on the function and its assumed level of complexity. The weights range from 1 for a simple operating system control language change to 12 for a very complex edit-validation function.

The value 3 is the most common among the 24 possible function-complexity assignments. If the function types are equally represented in programs, the average value is 4.

The programmer/analyst experience factor is an indication of the effect of experience on productivity. Values range from .75 to 2.75 corresponding to a lead analyst or programmer and interns respectively. Since experience is not evenly distributed over a group of programmers and analysts, the following groups was hypothesized in order to obtain an average or representative value for the experience factor.

Experience	Number in Group	Factor	Weighted Sum
Lead	1	.75	.75
Senior	2	1.25	2.50
Journeyman	4	1.75	7.00
Nominal	8	2.25	18.00
Intern	5	2.75	13.75
	20		42.00

Average Value = 42 ÷ 20 = 2.1

No definitions are provided for the 10 job classifications.

The job knowledge and turn-around time factors are self-explanatory.

The System Factor adjusts the product development effort to account for work already done. The product measure resulting from the format count and the program complexity value is the same whether the system is being developed in its entirety or it is a modification to an existing system. The system factor has the effect of modifying the product value to account for less than total development.

Seven levels of change are described by the System Factor. The values range from 2 for a new development to 8 for an operating systems control language change.

For a new system development the 2 in the primary estimating equation is divided by a System Factor value of 2 and the product measure is unchanged. Consequently, the System Factor values describing lesser amounts of new development have larger values and are portions of 2. The effect of the System Factor on the product measure is summarized as follows:

Type of Effort	System Factor	Effort Relative to a New Development
New Development	2	1.00
Major Change	3	.67
Major Modification	4	.50
Minor Modification	5	.40
Maintenance	6	.33
Minor Technical Change	7	.29
Operating Systems Control Language Change	8	.25

In order to get a feel for the relative magnitudes of the components of the Micro Estimating Procedure, consider the following example.

Number of I/O formats = 10

Number of functions = 20

Average complexity factor = 4.

New Development

Product = (Number of Formats + Weighted Number of Functions) x (Effort Related to a New Development)

Product = $(10 + 4 \times 20) \times 2 \div 2 = 90$

Experience = 2. (See above for computation)

Job knowledge required = 1.0

Job knowledge available = 1.0

Access = = 1.0

$(\text{Productivity})^{-1}$ = (Work Days per Unit of Product for a Staff with Average Experience) x (Job Knowledge Required) x (Job Knowledge Available) x (Access)

$= 2.0 \times 1.0 \times 1.0 \times 1.0 = 2.0$

Net Development Time = (Product) x $(\text{Productivity})^{-1}$

$= 90 \times 2.0 = 180$ Man-Days

If the effort was a major modification (System Factor = 4), the Product value becomes:

Product = $(10 + 4 \times 20) \times 2 \div 4 = 45$

and

Net Development Time = $45 \times 2.0 = 90$ Man-Days

If the Job Knowledge Required is "Detailed" (Factor = 1.5) and the Job Knowledge Available is "Limited" (Factor = 1.5), the productivity becomes:

$$(\text{Productivity})^{-1} = 2.0 \times 1.5 \times 1.5 \times 1.0 = 4.5$$

then for the major modification:

Net Development Effort = 45 x 4.5 = 202.5 Man-Days

Outputs

The primary output (i.e., the output that is sensitive or controlled by project variables as opposed to the subsequent step which is a fixed allocation) is: Gross Development Time (man-days). Gross Development Time includes:

- Nonproject time (individual assigned to project but busy with non-project tasks, e.g., training, non-product administrative duties, etc., and vacation and holidays)
- Wasted or lost time

Therefore, Gross Develpment Time describes the staffing level that will result in a needed amount of development time. The latter is predicted by program and project characteristics.

The secondary outputs (i.e., those derived by applying fixed values to the primary output are:

- Effort by project phase
- Total development cost

The project phases are:

- Review and analysis
- Design
- Programming
- Testing
- Documentation

Gross Development Time includes:

- Analysis of present methods
- Design of the new/changed system
- Develop the system's support
- Program design
- Program development
- Program testing
- System testing
- Installation and conversion
- Staff training
- Project officer
- System manager
- Technical managers
- Support personnel
- Documentation

Inputs

Product Related Inputs. The software is described by the numbers of types of items it processes and the numbers of functions it includes. The functions are described according to type and complexity. The result is two product descriptors: one measures the size of the input/output processing to be executed by the system; the other is a measure of the number and difficulty of the functions to be performed.

Input File Formats. The number of different formats to be read by the system are counted and added together. The model asks for numbers of card, tape, disk, and screen formats separately, but since the weighting factor is always one, there is no distinction made among them regarding the effort involved to implement them.

Output File Formats. The formats output by the system are totaled. The same entries as for the inputs are requested plus the number of report formats. As in the case of the inputs, the weighting factor for the different types of output is always one, so there is no reason to differentiate.

Program Complexity. The total program complexity measure is computed by a weighted sum of the number of processing functions of given types. Each function is characterized as simple, complex, or very complex. The processing functions are:

- Edit Validation
- Table Look-Up (Internal or External)
- Calculations
- Sort/Merge Process
- Internal Data Manipulation
- File Search
- Utilities or Subroutines
- Operating Systems Control Language

Job Knowledge Required. The amount of knowledge required to implement or change a system has a direct effect on the number of hours required to accomplish the project. A system that requires very detailed knowledge will require more effort than one that can be accomplished with limited knowledge. This parameter is paired with the job knowledge available factor described below to describe the relative influence on productivity. Three job knowledge levels are used: *Limited*, *General*, *Detailed*.

System Factor. The effort required to complete a system development or change project of given complexity depends on the state of the system. That is, the work required to change three file formats is less than the work required to develop a system with three file formats, all other factors being equal. The System Factor describes the level of effort being undertaken. Seven levels are described:

- System development
- Major changes
- Major modification
- Minor modification
- Maintenance
- Minor technical change
- Operating systems control language

Resource Related Inputs

Programmer/Analyst Experience Available. The available experience measure is an effective productivity indicator. It quantifies the rate at which the product can be produced in terms of the job classification of the staff available for assignment to the system development. Two data processing personnel classifications: Analyst and Programmer, are tabulated according to five levels of experience: Lead, Senior, Journeyman, Nominal, and Intern. Weights are associated with the difference experience levels. The result is a weighted average productivity factor.

Job Knowledge Available. This factor has the effect of describing the change in productivity associated with the level of knowledge about the work to be performed that exists among the persons available for assignment. It works together with the Job Knowledge Required factor described above to quantify the effect of the knowledge of the system required compared to that available on the time required to complete the work. In general, the effect of the combined factors is to increase the development manhours if the need exceeds the available and decrease the hours if the available exceeds the need. Three levels of job knowledge availability are specified: Limited, General, and Detailed.

Program Turn-Around Time. The effect of computer access on productivity is described by four levels of average turn-around time:

- Interactive terminal
- More than one run per day
- One run per day
- Less than one run per day.

Reference

Standard DoD ADP Resource Estimating Procedure (REP) for Software Development (Draft), Dept. of Defense, Sept 1979.

Doty Associates, Inc.

Description of the Model

The model is actually a set of 15 estimating relationships. Each one to be used for a given type of software and software life cycle phase. Equations have been derived empirically using regression analysis for the following types of software:

- Command and Control
- Scientific
- Business
- Utility

The development effort for software representing each of the application types may be estimated using one of three different relationships. An additional three are given that are applicable to all types of software. These equations are to be used "when the application cannot be categorized or is different than the categories noted". The procedure specifies that when a software system is made up of subsystems that are of different types, the total size should be divided into the four categories and the appropriate estimating equation used for each one. Then the individual manmonths are summed to give a total system development effort. The three equations are divided into size measure (lines of source code or words of object instructions) and the life cycle phase in which the estimate is made (Concept Formulation and all others). If the estimate is to be made using the words of object instructions, the same equation is used in all life cycle phases. Similarly, for estimating large systems (more than 10,000 lines) using lines of source code requires the use of a different equation in the Concept Formulation Phase than in the other life cycle phases.

The use of the different equations can be described as follows (A, B, and C refer to the three different relationships).

SOFTWARE DESCRIPTION	LIFE CYCLE PHASE	
	CONCEPT	OTHERS
WORDS OF OBJECT CODE	A	A
LINES OF SOURCE CODE		
LARGE SYSTEM $\geq$ 10K LINES	B	B
SMALL SYSTEM < 10K LINES	B	C

The forms of the estimating relationships are similar. Equations A and B are of the form:

$$MM = a\ I^{b}$$

where MM = Manmonths of development effort.

I = either words of object code (A) or lines of executable source code (B).

a,b = Constants obtained empirically.

Equation C has the form:

$$MM = c\ I^{d} \prod_{j=1}^{14} f_j$$

Where f_j = a set of parameters describing the development environment.

c,d = constants obtained empirically

Values of the constants to be used for different types of applications are given in Tables A-1 and A-2.

The following guidelines are presented for selecting the proper estimating relationship.

TABLE A-1

SUGGESTED UTILIZATION OF ESTIMATING RELATIONSHIPS FOR DEVELOPMENT MANPOWER

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] $MM = 2.060\ I^{1.047} \prod_{j=1}^{j=14} f_j$	⇨ ⇨
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] $MM = 0.501\ I^{1.263} \prod_{j=1}^{j=14} f_j$	⇨ ⇨
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] $MM = 2.011\ I^{1.019} \prod_{j=1}^{j=14} f_j$	⇨ ⇨
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] $MM = 3.742\ I^{0.781} \prod_{j=1}^{j=14} f_j$	⇨ ⇨
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 $I \geq 10{,}000$) $MM = 10.078\ I^{0.811}$ → TIME CONSTRAINED $MM = 10.078\ I^{0.811}$; TIME UNCONSTRAINED $MM = 1.426\ I^{1.243}$; (for $I < 10{,}000$)[1] $MM = 1.744\ I^{0.811} \prod_{j=1}^{j=14} f_j$	⇨ ⇨

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

TABLE A-2

SOFTWARE DEVELOPMENT MANPOWER ESTIMATING ALGORITHMS REFLECTING DEVELOPMENT ENVIRONMENT

FACTOR / ESTIMATOR FOR MM:		APPENDIX REFERENCE	APPLICATION									
			ALL		COMMAND & CONTROL		SCIENTIFIC		BUSINESS		UTILITY	
			$2.060\ I^{1.047}\ \prod_{j=1}^{j=14} f_j$		$0.501\ I^{1.263}\ \prod_{j=1}^{j=14} f_j$		$2.011\ I^{1.018}\ \prod_{j=1}^{j=14} f_j$		$3.742\ I^{0.781}\ \prod_{j=1}^{j=14} f_j$		$1.744\ I^{0.811}\ \prod_{j=1}^{j=14} f_j$	
		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	

- 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.*
- If accurate estimates of manpower requirements are required in the Analysis and Design and subsequent phases of development, use equation B, in source code, for programs of $I \geq 10{,}000$ and equation C, in source code, for programs with $I < 10{,}000$.
- For budgetary purposes, use the equation that gives the higher estimate.

Development time is estimated using the equation

$$D = \frac{1000I}{92.25 + 233I^{.667}}$$

where D = Reasonable development time in months

I = Number of delivered object instructions.

This relationship was obtained using regression on data describing 74 development projects. The time estimate should describe "customary" distributing of effort over time that is, it should avoid extremes of project time compression or expansion.

---

* according to one of the authors, size of the object code is recommended over size of the source code as an estimator because most of the developments of interest to the study sponsors are in the area of command and control and scientific systems. In these areas both estimators have similar precision, but the authors believe that object code is more commonly used and understood and is, therefore, a more reliable estimator.

It should be noted that a large portion of the documentation accompanying the description of the DAI estimating procedures is devoted to discussions of factors that are believed to influence the cost of software development. These factors are classified according to aspects of software and its development environment. The factors are grouped according to the following "domains":

- Requirements
- System Architecture/Engineering
- Management.

Outputs

Cost of Software Development

The estimate of total development cost is based on several relationships that portion the cost into components that can be estimated by applying available ratios to other costs and factors such as overhead and administrative costs. By the proper use of relevant values for these factors the relationships can represent either government in-house costs or contractor development costs. A method is described for time phasing the expenditure that is said to satisfy the requirements of DoD Directive 5000.1.

The procedure identifies costs that are incurred by the government during all phases of the software life cycle <u>except</u> Operation and Support. The total development cost includes:

$$C = C_{CF} + C_{VAL} + C_{FSD}$$

where

C = Development Cost
C_{CF} = Conceptual Phase Cost
C_{VAL} = Validation Phase Cost
C_{FSD} = Full Scale Development Cost.

Information is included that relates the government cost to the contractor's full scale development cost. This cost is the one developed by the formal software cost estimating procedure.

The cost of development is divided into primary and secondary costs, thus:

$$C_D = C_p + C_S$$

where C_D = Cost of Development
C_p = Primary Cost (Manpower)
C_S = Secondary Cost (Computer, Documentation, Etc.)

Then,

$$C_p = MM(C_e)$$

where MM = Total Development Man-Months
C_e = Average Labor Cost

and

$$C_S = \sum_{i=1}^{n} C_i = kC_p$$

Therefore: $$C_D = (MM)\ C_e\ (i + k)$$

where k = Ratio of Secondary to Primary Costs (=.075)

The total software development cost (does not include government Conceptual and Validation Phase costs) includes the costs of:

- Analysis
- Design

- Code
- Debug
- Test and Checkout

and is proportional to the total man-months of development effort.

Total Development Man-Months

This is the primary output variable. It is the basis for the total development cost estimate and it is the value from which the distribution of effort by life cycle phase is derived. The hours include those directly related to the development of the software system. They include the direct hours needed for:

Analysis - interpreting the system requirements and producing viable alternative system concepts

Design - preparing detailed designs of the data processing system and the individual programs

Coding and Debugging - writing individual modules and programs and performing individual tests

Testing and Checkout - integrating the individual subsystems into a complete system and conducting prescribed tests on the entire system.

The discussion of the model does not indicate the extent that support and management hours are included in the total. Also, there may be some question about the activities associated with concept development (e.g., is the test plan furnished by the government following the validation phase or is it developed as part of the project). As in many cost estimating situations, the line between concept analysis and the evaluation of solutions to selected concepts is hazy.

Although the DAI documentation and discussions with the authors indicate that the model includes integrated system testing, it appears that this effort is not included in the original SDC data which was the basis for the curve fits. (76% of the SDC data points describe programs that do not interface with any other programs).

Software Development Time

A nominal development time is presented that implies "customary manloading". That is, the schedule does not reflect either crash projects or allow for unnecessary delays.

Distribution of Development Effort

The expenditure of time and effort associated with major project milestones is given for small projects (one level of supervision) and large projects (more than one level of supervision). The distributions are for nominal projects and do not allow for any possible acceleration or delay of the completion of the project.

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%

Inputs

Program Size

DAI has been very careful to describe the size variables which are the primary inputs to the estimating equations. This should help make more reliable estimates using the relationships. However, we should point out that the respondents to the original SDC questionnaire were not so well directed and it may be necessary when analyzing the structure of the model as it relates to prediction accuracy that significant errors may have been introduced by this failure to be specific. The DAI model may not overcome what are inherent limitations in the data.

The DAI procedure calls for several estimates in support of the DSARC process. It recognizes that the best estimates of program size are obtained later in the development cycle. It suggests, then, that the interpretation of the program size changes during the life cycle and that associated with the changes are increases in estimating accuracy. The report describes how the knowledge of the size estimator changes during the life cycle and how this affects the estimating precision. The precision associated with the different size measures during the system development life cycle is as follows.

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.

Code that is developed as part of the project but is not delivered to the customer is a source of variation in the estimate of the system size and must be considered. However, no guidance is provided for making any adjustment other than citing that the SDC data showed delivered code to average 77 percent of the developed code with a standard error of 30 percent.

Allowance must also be made for support software development especially when working with new hardware.

Total Object Words

During the Conceptual Phase when very little is known about the system to be developed, the initial estimate is made using the analyst's judgement (usually by analogy with previously developed systems, but other methods are possible) of the number of object words occupied by "every program needed to run and maintain the system in the field". This measure is obtainable from listings of computer system routines that build executable programs from the output of the compiler. Taking values from systems similar to the one being planned can provide a basis for estimating the value. Care should be taken, however, when program overlays are involved. Also, extensive use of standard library routines can greatly increase the words of object program size and not be representative of a comparable increase in development effort.

Total Object Words Minus Data Areas

The memory space occupied by an executable program is composed of locations containing instructions and locations reserved for the data upon which the program will operate. Sometimes the data storage areas are significantly larger than the area occupied by the actual instructions. DAI suggests that the effort required to develop the programs is more closely related to the size of the instruction space than to the size of the combined data and instruction storage. However, as in the case of the total object words, there is no evidence of this distinction being made

in the original derivation of the estimating procedures. Also, there is no guidance provided on how to apply the additional information when preparing cost estimates. Some computer system executive processing routines provide this information. However, many don't and, therefore, it would be very difficult to obtain comparable historical information to guide new estimates.

New Object Words Minus Data Areas

Only the writing of new code contributes to the software development effort (if code written to modify existing modules is counted as new code). To account for the work done to adapt existing code to a new system, which includes analyzing the code and deciding how to modify it, any existing module that will result is less than 50 percent utilization of existing code is considered to be entirely new.

New Source Lines

Counts of new source lines written (whether in a higher order or machine oriented language) can be obtained from compiler listings, measuring card decks or text editors. It is one of the easiest measures of size to obtain. As in the previous case, modules containing less than 50 percent reused code are considered to be new.

Development Environment

For estimates made using lines of source code where the size is less than 10,000 lines, the estimating relationship includes a number of factors describing the development environment. These are included in the estimate when the indicated item is to be part of the development process. Detailed definitions of the factors are presented in an Appendix.

f_1 Special Display
f_2 Detailed Definition of Operational Requirements
f_3 Change to Operational Requirements
f_4 Real Time Operation

f_5 CPU Memory Constraint
f_6 CPU Time Constraint
f_7 First SW Developed on CPU
f_8 Concurrent Development of ADP Hardware
f_9 Time Share Versus Batch Processing in Development
f_{10} Developer Using Computer at Another Facility
f_{11} Development at Operational Site
f_{12} Development Computer Different from Target Computer
f_{13} Development at More than One Site
f_{14} Programmer Access to Computer

After analyzing the method used by DAI to obtain their estimating relationships and after comparing their definitions of input and output variables with the original sources of data, it is clear that there are discrepancies between the way the data are being applied and what they originally represented. DAI does not explicitly justify their approach but their presentation of the estimating procedure does give consideration to errors arising from differing definitions of the variables.

DAI seems to be saying that consistent use of the estimating procedures regardless of how they were obtained will produce results with at least a predictable error. That is, knowing the range of error that can occur because of differences in definitions and ability to predict the input variables will, when applied to the given estimating relationships, produce estimates with precision that is in accordance with previous experience. DAI further substantiates the approach of throwing all the error into the ability to define the input by presenting standard error values for the size variables at different times in the life cycle.

References

J. H. Herd, J. N. Postak, W. E. Russell, K. R. Stewart, Software Cost Estimating Study, Study Results, Vol. I, NTIS AD A042264, Jun 1977.

D. L. Doty, P. J. Nelson, K. R. Stewart, Software Cost Estimation Study, Guidelines for Improved Software Cost Estimating, Vol. II, NTIS AD A042264.

FARR AND ZAGORSKI MODEL

Description of the Model

System Development Corporation completed several projects for the Air Force, Electronic Systems Division in which they attempted to develop methods for predicting the cost of software development. The Farr and Zagorski model represent an intermediate stage in the program.

Using historical data from internal projects and from other organizations, the SDC team systematically tested over 100 variables to learn if they were satisfactory predictors of program design, coding and debugging effort.

Farr and Zagorski published three equations* which were determined to be the best predictors tested up to that time.

$$MM = 2.7X_1 + 121X_2 + 26X_3 + 12X_4 + 22X_5 - 497 \qquad (1)$$

$$MM = 2.8X_6 + 1.3X_7 + 33X_3 - 17X_8 + 10X_9 + X_{10} - 188 \qquad (2)$$

$$MM = 8.4X_{11} + 1.8X_{12} + 9.7X_3 - 3.7X_{13} - 42 \qquad (3)$$

Definition of Output

MM is the number of manmonths needed to design, code and debug a single program. The effort begins when a programmer or analyst is given a complete operational specification for a program and it ends when the program is released for integrated system testing.

Definitions of Inputs

X_1 = number of instructions in original estimate (in thousands)

X_2 = subjective rating of information system complexity (scale 1-5)

X_3 = number of document types delivered to customer

X_4 = number of document types for internal use

* L. Farr and H. J. Zagorski, "Quantitative Analysis of Computer Programming Cost Factors: A Progress Report," Proceedings ICC Symposium, Rome, 1965

X_5 = number of computer words needed to store program data ($\log_{10}$)

X_6 = number of instructions in delivered program (in thousands)

X_7 = number of man-miles for travel (in thousands)

X_8 = system programmer experience (average of total years of experience with the computer, language, and application)

X_9 = number of display consoles

X_{10} = percent of instructions new to this program (not re-used from previous versions)

X_{11} = number of instructions to perform decision functions (in thousands)

X_{12} = number of instructions to perform nondecision functions (in thousands)

X_{13} = programmer experience with this application (average number of years).

References

L. Farr, H. J. Zagorski, Factors that Affect the Cost of Computer Programming, Vol II, A Quantitative Analysis, NTIS, AD607546, Sep 1964. Quantitative Software Models, Data and Analysis tenter for Software, SRR-1, Mar 1979

PRICE S

Description of the Model

PRICE S is a proprietary software cost estimating model developed and maintained by PRICE Systems Division of RCA, Cherry Hill, New Jersey. It is installed on the On-Line Systems, Inc. time sharing network and may be accessed using several different types of terminal devices. On-Line Systems provides local dial-up service in many cities throughout the United States.

PRICE S estimates software development costs by systematically adjusting the estimate of an initial element of the cost structure. The initial estimate is a function of the size of the system to be developed and several other parameters describing the characteristics of the software and the development environment.

According to Frank Freiman, the creator of PRICE S, the model design reflects an understanding of why costs attain their values. This is in contrast with estimating costs by fitting hypothesized relationships to historical data. The PRICE S developers contend that software development projects are complex undertakings whose costs are influenced by a multiplicity of factors too numerous to analyze and often impossible to measure. Consequently, no two projects are alike. This makes it impossible to identify common characteristics among past projects to be used for estimating new ones. It also means that no past project is exactly relevant to any future one.

Freiman contends that a manager's perception of what a project should cost actually determines its estimated cost. Therefore, the proper formulation of a cost estimating model is in terms of perceived relationships between cost and aspects of software development that knowledgable managers believe determine cost. PRICE S estimates the cost of developing code using hypothesized relationships that were subsequently supported by the opinions of individual managers. The primary relationships describe:

- The cost required to produce programs.
- The effect on cost of changing development time.
- The comparative costs of the development cycle elements.

In addition to the primary relationships there are many secondary ones which may be subjective or empirical. The ultimate justification for all the hypotheses is the model's estimating performance.

The relationship between the cost of producing a program and attributes of the code was originally conceived using an analogy. The analogy is between the force required to move an object and the effort needed to write software. The mass of an object is the product of its volume and density. The property of mass is a function of its spatial concentration and its extent or volume. In a similar manner the cost or producing code is related to the product of its density or difficulty and its extent or size. Furthermore, the cost relationship should satisfy the empirical observation that the cost per pound of a wide range of items decreases as the weight increases.

In PRICE S the difficulty associated with a computer program is represented by a parameter called its APPLICATION (APPL). The value of APPL is small for easy to write codes such as mathematical applications (APPL=0.86) and is large for interactive operations and operating systems (APPL=10.95).

The size of the system is measured in terms of the number of machine level instructions represented by the developed code. PRICE S uses the mnemonic INST to represent the size of the software system.

The product (APPL) X (INST) is termed the weight or mass of the system and represents a portion of the development effort.

For a given system the development effort is affected by the development time as well as the type of application. A system with an accelerated schedule becomes more difficult in the same sense that a system with a greater value of application is more difficult.

The APPL values that are input to the model are related to a standard schedule of 9 months. The model compares the schedule for the project being estimated with the standard schedule and calculates an effective value of APPL which is used by the model to make estimates. The effective value of APPL is not available to the user.

The model does not require that the user specify the development schedule. This is possible, but the reference manual recommends that the schedule be calculated by the model using the following procedures.

A project with given size, application type, scope of work, etc. may be developed with different schedules. The effect on cost of changes in schedule is represented by the parameter called COMPLEXITY (CPLX) ". . . which provides a quantitative description of the relative difficulty of the design task."* CPLX describes the familiarity of the project staff with the functions to be performed, their general experience, and factors that complicate the development of the system such as new language, more than one user organization, or state-of-the-art advancement. CPLX tends to be constant for a given organization. It reflects the way the organization commits its resources in order to achieve a perceived proper scheduling of a project. If CPLX is input to PRICE S, the model calculates the schedule. If the schedule is given, the model calculates CPLX. If both are given, the model calculates the schedule according to the value of CPLX and makes adjustments in cost depending on whether the stated schedule represents an acceleration or deceleration of the first schedule.

In any case the resulting schedule is compared with the 9 month standard schedule to obtain the effective value of APPL as described earlier

* Reference Manual PRICE Software Model, RCA/PRICE Systems, Cherry Hill, New Jersey

The initial estimate of cost in PRICE S is the Engineering Design element. It is obtained from the relationship:

Engineering Design Cost = (WEIGHT) X (RESOURCE)

RESOURCE (RESO) is the cost per pound mentioned above. It represents the efficiency with which an organization uses its resources to develop a system. Its value should be constant for an organization. The value is obtained by operating the model in a calibration mode which calculates RESO from data describing past projects.

RESO and CPLX act together to describe an organization. RESO measures organizational attributes that affect cost independently of schedule, while CPLX measures those attributes that affect schedule.

Other parameters than the ones described so far are used to calculate cost. These will be described later under the description of inputs. The primary purpose of this presentation of the model is to describe the general model structure and the principal inputs. Figure A-1 describes the sequence of the calculations and the model variables.

The calculation of the cost elements follows from the initial estimate of Systems Engineering Design Cost using a sequence of allocations called the "Ripple Effect." These are shown in Table A-3. These allocations can be modified by user inputs. The cost elements are defined in the discussion of outputs.

Outputs

PRICE S offers a number of operating modes. Many of these modes involve tailoring or constraining the development process to satisfy user requirements. In these cases the normal outputs of the model may become inputs. The following presentation will assume that the standard estimating situation is the description of software and unconstrained resources resulting in model estimates of cost and schedule. Specified values and constraints will be treated as special cases.

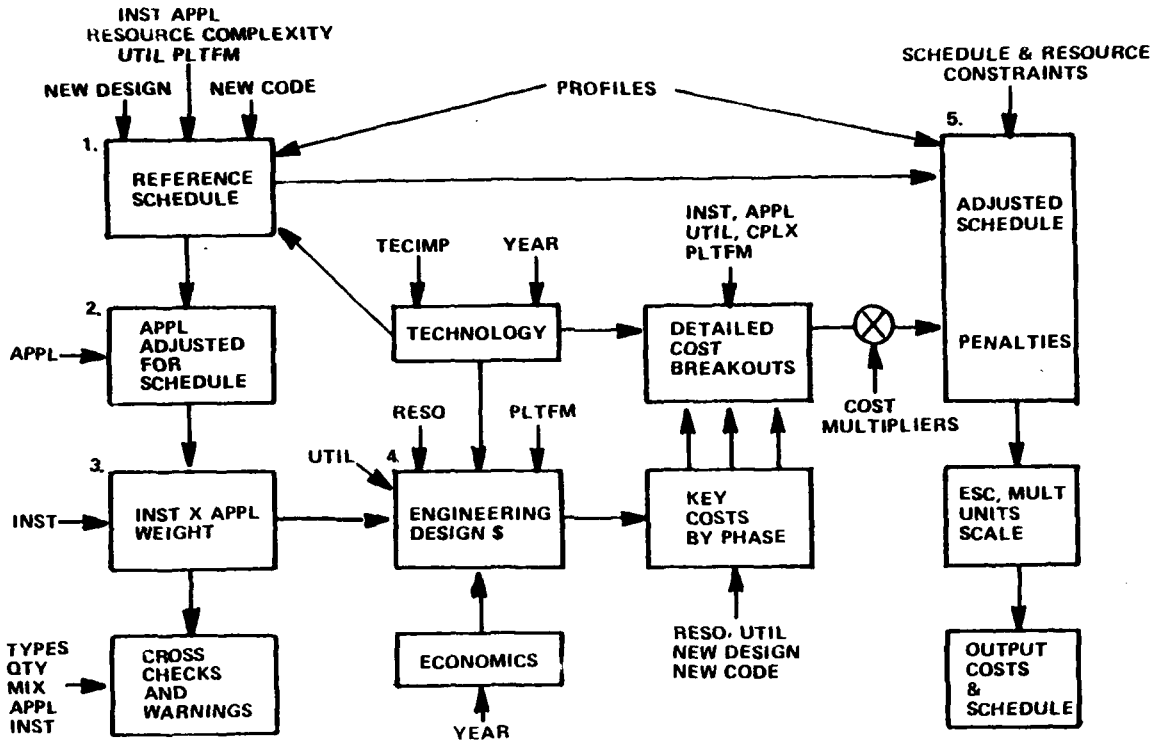
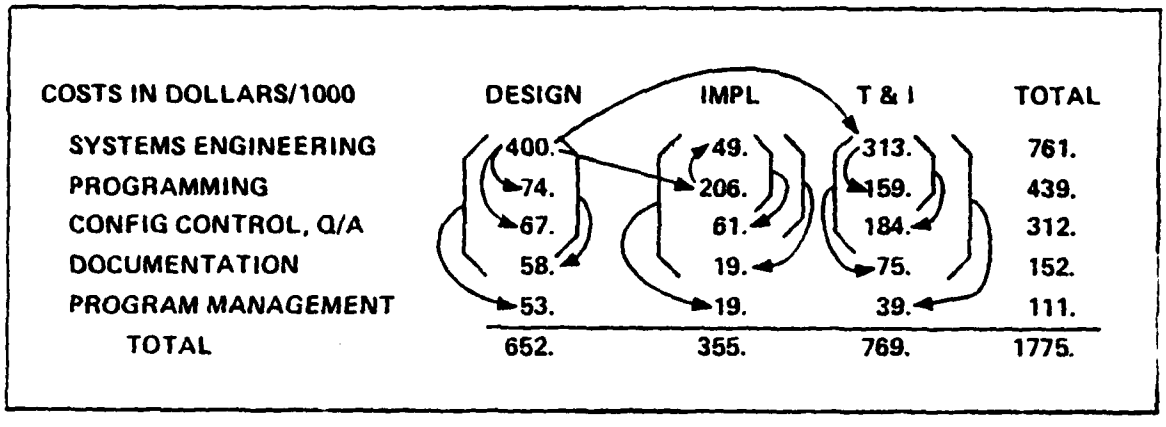


Figure A-1 Sequence of Calculations in PRICE S

TABLE A-3

SOFTWARE PROGRAM COSTS RIPPLE EFFECT



COSTS IN DOLLARS/1000	DESIGN	IMPL	T & I	TOTAL
SYSTEMS ENGINEERING	400.	49.	313.	761.
PROGRAMMING	74.	206.	159.	439.
CONFIG CONTROL, Q/A	67.	61.	184.	312.
DOCUMENTATION	58.	19.	75.	152.
PROGRAM MANAGEMENT	53.	19.	39.	111.
TOTAL	652.	355.	769.	1775.

Development Schedule

PRICE S divides the software development cycle into three phases:

- Design
- Implementation
- Test and Integration

The beginning and ending month and year is given for each phase along with a bar graph representation of the schedule. The phases are allowed to overlap in time.

The Design Phase begins with the design of the system to be developed under the project. Activities in the Design Phase include:

- Establish system architecture
- Allocate system requirements to programs
- Design programs in detail

When each program design is completed, coding can begin in that program. If it is necessary to change the program design, the activity of design is considered part of the design cost even if coding has begun. Although this is a desirable distinction, the definition of the programming activity (see below) does not permit the user to know how much redesign cost may be included in an estimate.

The Implementation Phase is devoted to writing the program code and debugging the individual programs. Under the development concept reflected in the model, programs are designed, coded and debugged as individual units. Therefore, the Implementation Phase begins when coding starts on the first program to complete the design process and continues until the last program is ready for formal testing.

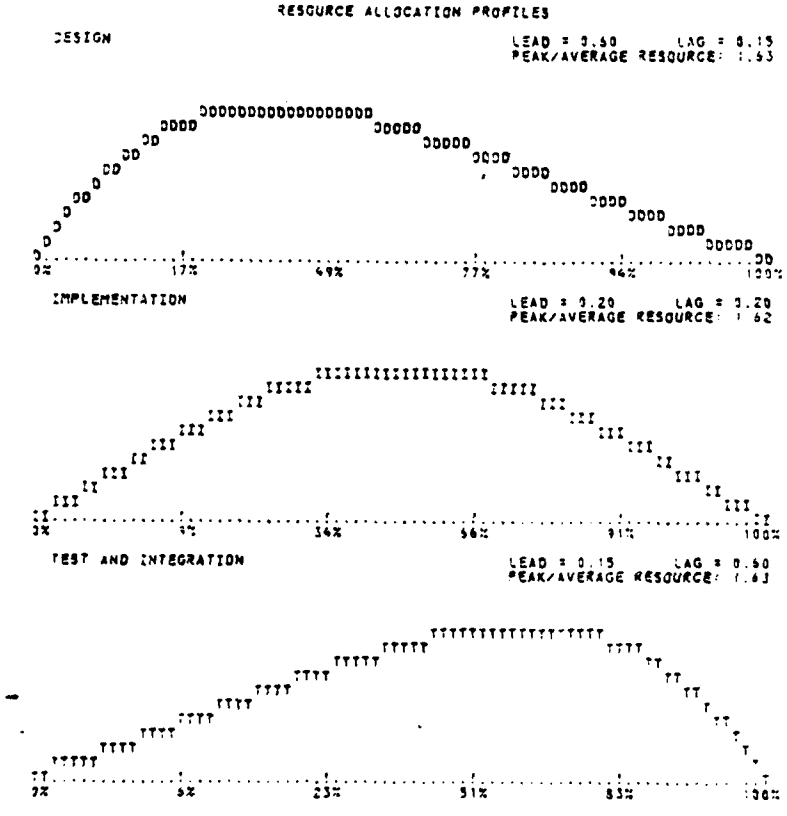
Implementation Phase activities include:

- Program coding
- Program debugging
- Program documentation

The Test and Integration Phase begins with the test planning activity. Therefore, it can start before any coding begins. It extends until the system is accepted by the user. The major activities include:

- Test planning
- System construction from individual programs
- Program testing
- System testing

The overall development schedule is obtained from the user-specified start date (DSTART) and the system complexity (CPLX). In the absence of any schedule constraints the model calculates a nominal schedule. The calculation of resource distributions among the three phases and the overlap are performed using Beta functions.



The shapes of the three profiles can be changed by the user. PRICE S adjusts the overlap to obtain a smooth shape of the total resource curve over time.

Development Cost, Constant Dollars

Development Cost is given by phase and activity. The phases (Design, Implementation, and Testing and Integration) are described above. Each phase is divided into 5 activities or cost elements:

- System engineering
- Programming
- Configuration control and quality assurance
- Documentation
- Program management

System Engineering is the technical direction of the system development. It includes the following tasks:

- Development of system specifications
- Allocation of the system functions to programs
- Description of program interfaces
- Evaluation of system performance
- Problem resolution

The Programming activity includes design and coding, and testing individual programs. These three tasks are normally performed in the three corresponding phases, but as was stated above design can occur in the Implementation Phase. Other activities can also occur in different phases.

Documentation includes:

- Draft preparation
- Editing
- Reproduction
- Distribution
- Review
- Revision

Configuration Management is the control of the description of the approved system. The three principal tasks are:

- Defining the system baseline
- Managing the process for changing the baseline
- Disseminating information describing the system

Program Management includes ". . . the supervisory, financial, legal, and general administrative tasks necessary to plan, organize, direct and control the project." *

The preceding definitions along with the definitions of the development phases are presented to give an idea of what the PRICE S creators consider to be the principal cost elements of a software development project. However, the nominal allocations of the costs by element (see the ripple calculation in the Description of the Model) can be changed by the user to suit his own definitions. The ability to reallocate costs when exercised along with the calibration of the model using cost values representative of a given organization make it possible for the user to define the model cost elements almost any way he pleases.

Costs are calculated using a fixed reference year (1976 for the version of PRICE S tested) and corrected for inflation to the first of the year in which the project start date occurs. An internal table of inflation rates (RTABLE) accomplished the adjustment of value. The standard table can be changed by the user if desired. Under the constant dollar option, the costs are in base year dollars.

* *Op. Cit.*

PRICE S provides an option for presenting costs in other currencies An input value establishes the conversion rate between dollars and the other currency.

Development Cost, Inflated Dollars

PRICE S includes an option that allows the user to obtain all cost elements in terms of an inflated currency. Dollars (or another specified currency) are converted from the base year to time during the development using a table of constants (RTABLE) that is either specified by the user or obtained by default. The effect of inflation on each cost element is determined by the Beta function distribution of resources over time (see above).

Development Effort

An option is available under which PRICE S presents the cost elements in terms of effort rather than currency. The user can select either man-months or man-hours. All reports are appropriately labeled. Since the internal calculations are in terms of dollars, the user must specify the cost per man-month to effect the conversion to the desired output. The effects of inflation on effort are obtained as in the case of reporting cost in currency.

Model Output Options

PRICE S incorporates many execution options and provides for many alternative presentations of the output data. The model allows the user a great amount of flexibility in stating his inputs and obtaining reports. It should be noted, however, that these diverse representations need not be exercised if the user elects to use the default values provided by the model. The following are a few of the model output options.

Normal Output

The default report written by PRICE S includes information describing the software and project parameters, cost and schedule. A sample

report is shown in Figure A-2. PRICE S includes print control options that present specific portions of the information in Figure A-2.

Sensitivity Option

The effect on cost of changing the values of four variables can be conveniently shown in a 3 X 3 matrix. Under one option RESOURCE and COMPLEXITY are given nominal, higher and lower values and the development costs for the nine pairs are calculated. Another option performs the same calculations for the APPLICATION and INSTRUCTIONS inputs. Under either option the user may let the model set the ranges of the input values or he may specify them himself. The options offer a convenient way to examine how uncertainty in the most important inputs affects the estimated development cost. Figure A-3 shows the matrices printed using the sensitivity options.

Schedule Option

If the user specifies the development schedule, this option compares the user's schedule with the model's normal schedule and prints the effect on the development cost of departing from the normal schedule. It is axiomatic in the PRICE S model that either increases or decreases from the normal schedule have the effect of increasing the development cost.

Curve Option

The model prints a monthly history of the effort and cost. A cumulative percent completion is reported for each phase. Cost is given as monthly and cumulative values and their related percentages.

Design-to-Cost Option

Given values of the target cost, APPLICATION, RESOURCE, and COMPLEXITY, PRICE S calculates the size of the largest system that can be built.

--- PRICE SOFTWARE MODEL ---

DATE 29-AUG-79 TIME 09:47 (790239) FILENAME:S3

SAMPLE BOX 1 SAMPLE FILE

DESCRIPTORS					
INSTRUCTIONS	36000	APPLICATION	5.30*	RESOURCE	3.50
UTILIZATION	0.50	PLATFORM	1.40	COMPLEXITY	1.25
NEW DESIGN	0.98*			NEW CODE	0.99*

COSTS IN DOLLARS/1000	DESIGN	IMPL	T & I	TOTAL
SYSTEMS ENGINEERING	375.	40.	244.	659.
PROGRAMMING	69.	191.	122.	382.
CONFIG CONTROL, Q/A	62.	56.	149.	268.
DOCUMENTATION	54.	17.	59.	131.
PROGRAM MANAGEMENT	50.	17.	30.	97.
TOTAL	611.	322.	604.	1537.

SCHEDULE AND CONSTRAINTS	DESIGN	IMPL	T & I
START WORK	OCT 80	FEB 81*	MAY 81*
END WORK	JUL 81*	DEC 81*	AUG 82*
COST PER MAN-MONTH(1980 DOLLARS)	0.0	0.0	0.0
MAXIMUM MAN-MONTHS PER MONTH	0.0	0.0	0.0

APPLICATION CATEGORIES	MIX	NEW DEVELOPMENT DESIGN	CODE	HARDWARE INTERFACES TYPES	QUANTITY
DATA S/R	0.0	0.0	0.0	0	0
ONLINE COMM	0.08	1.00	1.00	1	1
REALTIME C&C	0.08	1.00	1.00	2	2
INTERACTIVE	0.23	1.00	1.00	1	2
MATHEMATICAL	0.28	0.50	0.70	***	***
STRING MANIP	0.26	1.00	1.00	***	***
OPR SYSTEMS	0.07	1.00	1.00	***	***

SIZING DATA					
FUNCTIONS	3	STRUCTURE	0.0	LEVEL	0.0
CAPACITY	0	SOURCE	5696*	EXPANSION	6.32

SUPPLEMENTAL INFORMATION					
YEAR	1980	MULTIPLIER	1.000	ESCALATION	0.0
TARGET COST	0.	INTEGRATION	0.50	ESC EFFECT	1.00*

SCHEDULE GRAPH

OCT 80 AUG 82

********** DESIGN **********

********** IMPLEMENT **********

********** TEST & INTEGRATE **********

Figure A-2. Standard PRICE S Cost Report

--- PRICE SOFTWARE MODEL ---

DATE 29-AUG-79 TIME 15:51 (790239) FILENAME:S3

COSTS IN 1980 DOLLARS/1000

COMPOSITE SENSITIVITY DATA
(RESOURCE - COMPLEXITY)

RESOURCE CHANGE	COMPLEXITY CHANGE -0.100	COMPLEXITY CHANGE 0.0	COMPLEXITY CHANGE +0.100
-0.100	COST 4308. MONTHS 29.7	COST 4681. MONTHS 32.3	COST 5057. MONTHS 34.9
0.0	COST 4497. MONTHS 29.8	COST 4889. MONTHS 32.4	COST 5282. MONTHS 35.0
+0.100	COST 4690. MONTHS 29.9	COST 5099. MONTHS 32.5	COST 5510. MONTHS 35.1

--- PRICE SOFTWARE MODEL ---

SAMPLE BOX 1 SAMPLE FILE

COSTS IN 1980 DOLLARS/1000

SENSITIVITY DATA
(APPLICATION - INSTRUCTIONS)

APPLICATION	INSTRUCTIONS 32400	INSTRUCTIONS 36000	INSTRUCTIONS 39600
5.199	COST 1362 MONTHS 21.4	COST 1510. MONTHS 22.2	COST 1658. MONTHS 23.0
5.299	COST 1386. MONTHS 21.5	COST 1537. MONTHS 22.4	COST 1688. MONTHS 23.2
5.399	COST 1410. MONTHS 21.6	COST 1564. MONTHS 22.5	COST 1718. MONTHS 23.3

Figure A-3. Sensitivity Analyses

System Integration and Test Option

This operating mode calculates the cost associated with a system that is composed of independently developed parts. The cost of such a development is greater than the sum of the costs of the pieces. Additional costs are incurred for defining and maintaining the specifications of the subsystems and their interfaces. There are also costs associated with integrating the subsystems into the total system and conducting total system tests.

The amount of integration and test cost is determined by a single input value (INTEG) for each subsystem. The value of INTEG ". . . relates the level of engineering, programming and testing effort involved to integrate the subsystem into the total unified operation."* It takes on values between 0 and 1.

The System Integration and Test Option is unique among the models tested because it presents costs for the individual subsystems as well as the total system with the added cost of integration.

Verification and Validation Option

PRICE S calculates the cost of independent verification and validation of the new system using values of INTEG between .7 and .8 and proceeding as above.

Test Bed Option

The cost of installing the new system on a computer other than the one which it was developed is estimated. The model performs cost calculations based on the assumption that installation on a new computer involves redesigning and recoding a small part of the code. Ten percent redesign and rewrite is considered representative.

* Op. Cit.

Other Options

PRICE S has many additional output options including:

- Condensed cost and project reports (4 options)
- Subsystem level reports
- Model constants report
- Inflation rate table
- Cost multiplier table
- Effort distribution constants
- Sensitivity constants
- Resource allocation profiles
- Namelist table

Inputs

Software development projects and their environments can be described by as many as 64 constants and 4 tables (Figure A-4). The use of some of the inputs excludes some of the others and most of the parameters have default values provided by the model. Therefore, the user may describe a given development effort using different but equivalent inputs (e.g., number of Object Instructions or number of Source Statements and Expansion Ratio) and at different levels of detail (e.g., an assumed value of RESOURCE instead of a description of MIX). It is possible to execute the model by specifying only 8 values:

- INST
- APPL
- RESO
- UTIL
- PLTFM
- CPLX
- YEAR
- MULT

The following discussion explains each of these quantities, and selected others that serve to describe the model's ability to define software and the development environment.

Project Title									
Project Category									
Descriptors	INST	APPL	RESO	UTIL	PLTFM	CPLX	NEWD	NEWC	
Schedule	DSTART	DEND	ISTART	IEND	TSTART	TEND			
Resource Constraints	DCOST	DMAX	ICOST	IMAX	TCOST	TMAX			
Mix	MDAT	MONL	MREA	MINT	MMAT	MSTR	MOPR	MAPPS	APPLS
New Design	DDAT	DONL	DREA	DINT	DMAT	DSTR	DOPR	DAPPS	
New Code	CDAT	CONL	CREA	CINT	$CMAT_g$	CSTR	COPR	CAPPS	
Interface Types	TDAT	TONL	TREA	TINT					
Interface Quantities	QDAT	QONL	QREA	QINT					
Sizing Data	FUNCT	STRU	LEVEL	CAP	SOURCE	EXPAN			
Supplemental Information	YEAR	MULT	ESC	TARCST	INTEG				
Program Constraints	GTABLE=								

Notes:

Note: Shaded Areas Indicate Optional Inputs Used To Refine Or Modify The Basic Input Set

Figure 4-4 PRICE S Inputs

System Size

The size of the software system produced by the development project is stated in terms of INST, ". . . the total number of delivered, executable, machine-level instructions. Comments, format statements and data declaration statements should, in general, not be included." *

Delivered instructions limits the size measure to programs that are turned over to the customer. This would exclude special development programs, file conversion routines or test drivers.

Executable instructions are those that involve computer operations in contrast with data and constant storage.

Machine-level instructions are the most elementary operations of the computer. Each one may require from one to several words of primary storage. PRICE S operates internally using the above definition of size. However, the model offers two alternative size measures. The first is in terms of number of source instructions and an expansion ratio; the second uses the number of system functions; and the third alternative uses the system logic structure.

The selection of source statements (SOURCE) and expansion ratio (EXPAN) as the system size measure offers some flexibility in the definition of source program size. It is necessary only that the product of the expansion ratio and the source program measure approximate the number of delivered executable machine-level instructions. Therefore, the expansion ratio can include an allowance for comments and data storage instructions as long as the proportion of these to the total number of statements is relatively constant. The number of machine-level instructions is estimated by the product of the number of source statements and the expansion ratio:

$$INST = SOURCE \times EXPAN$$

* Op. Cit.

The second alternative size measure uses the product of the number of functional modules to be included in the software development (FUNCT) and the average number of machine instructions per function (INSPF):

$$INST = FUNCT \times INSPF$$

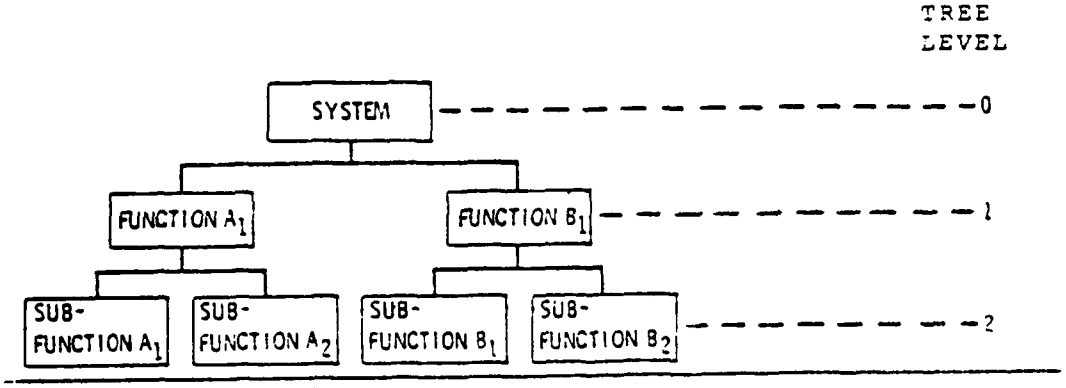
INSPF is a table entry with a programmed value of 90. The user may specify a different value.

The third method for specifying size uses an empirically derived variable (STRU) to relate the number of functional modules (FUNCT) and the average functional level of the system (LEVEL):

$$FUNCT = STRU\ (1 + LEVEL)^{1.2} + STRU$$

PRICE S will calculate values of STRU given FUNCT and LEVEL from past projects. The values obtained can be used to make new estimates.

LEVEL is obtained from the functional tree diagram (Figure A-5).



COMPUTATION:

TREE LEVEL	NO. OF FUNCTIONAL MODULES	PROQUCT
0	1	0
1	2	2
2	4	8
SUM	7	10

$$LEVEL = \frac{PRODUCT}{SUM} \cdot \frac{10}{7}$$

$$= 1.429$$

Figure A-5. Computation of LEVEL

LEVEL is a weighted average with the weights being the number of functions at each level of the tree. LEVEL, like STRU, can be calculated from projects similar to the one being estimated.

Once a value of FUNCT is obtained, INST is calculated as before by using INSPF.

Application

PRICE S uses this parameter to characterize the difficulty of the programming task. It is intended to adjust the relationship between cost and program size to account for the inherent differences in resources associated with different types of applications. The application parameter (APPL) ". . . represents an inherent instruction complexity, independent of variation in resources, schedules, operating environment and system utilization."*

Acceptable values of APPL range from 0.866 to 10.952. The lower end of the range is associated with programs that are predominantly math and string manipulations; the higher values represent real-time command and control and interactive applications. Increasing values of APPL describe programming tasks that require more resources for a system of given size.

Values of APPL for a given estimating situation may be assigned on the basis of experience with similar systems. Or, APPL for the system, may be calculated from a weighted sum of its component parts. This alternative determination of APPL is obtained by estimating the proportion of the total system size represented by each of seven categories of automated functions (Mix Categories, Table A-4). The model lets the user define his own category and APPL value if necessary.

* Op. Cit.

TABLE A-4

MIX Categories

MIX CATEGORY	IDENTIFYING CHARACTERISTICS
DATA STORAGE AND RETRIEVAL: (MDAT) APPL=4.10	-OPERATION OF DATA STORAGE DEVICES -DATA BASE MANAGEMENT -SECONDARY STORAGE HANDLING -DATA BLOCKING AND DEBLOCKING -HASHING TECHNIQUES -HARDWARE ORIENTED
ON-LINE COMMUNICATIONS: (MONL) APPL=6.16	-MACHINE-TO-MACHINE COMMUNICATIONS WITH QUEUING PERMITTED. -TIMING REQUIREMENTS NOT AS RESTRICTIVE AS WITH REAL TIME COMMAND AND CONTROL
REAL TIME COMMAND AND CONTROL: (MREA) APPL=8.46	-MACHINE-TO-MACHINE COMMUNICATIONS UNDER TIGHT TIMING CONSTRAINTS -QUEUING NOT PRACTICABLE -HEAVY HARDWARE INTERFACE -STRICT PROTOCOL REQUIREMENTS
INTERACTIVE OPERATIONS: (MINT) APPL=10.95	-MAN-MACHINE INTERFACES -HUMAN ENGINEERING CONSIDERATIONS -ERROR DETECTION AND PROTECTION
MATHEMATICAL APPLICATIONS: (MMAT) APPL=0.86	-ROUTINE MATHEMATICAL APPLICATIONS WITH NO OVERRIDING CONSTRAINTS
STRING MANIPULATION: (MSTR) APPL=2.31	-ROUTINE APPLICATIONS WITH NO OVERRIDING CONSTRAINTS -NOT ORIENTED TOWARD MATHEMATICS -TYPIFIED BY LANGUAGE COMPILERS, SORTING, FORMATTING, BUFFER MANIPULATION, ETC.
OPERATING SYSTEMS: (MOPR) APPL=10.95	-TASK MANAGEMENT -MEMORY MANAGEMENT -HEAVY HARDWARE INTERFACE -STRICT TIMING REQUIREMENTS -HIGH RELIABILITY

Given the proportions of code in each category the system application is calculated as follows:

$$APPL = \sum_{i=1}^{8} (MIX_i)(APPL_i)$$

where MIX_i = the proportion of the system code in the ith MIX category

$APPL_i$ = the APPL value for the ith category (see Table A-4).

Resource

RESO represents the effects on cost of items such as:". . . skill levels experience, productivity, efficiency, computer operating charges, and labor and overhead rates of the organization."* The PRICE S estimating procedure reflects the assumption that this value remains fixed in an organization. The value of RESO is obtained from historical data using the PRICE S calibration mode.

A large organization that includes many separate groups may present different values of RESO. In making cost estimates for such organizations it would be necessary to ascertain that the RESO value is consistent with the particular group that will undertake the project being estimated.

Utilization

UTIL describes the proportion of available computer memory occupied by the application programs. It also describes the fraction of the computer cycle time required to execute the program.

The PRICE S Reference Manual does not describe how combined time and space constraints are represented by UTIL. Discussions with PRICE S staff members suggest that the parameter represents a subjective assessment of the effect of either one or both types of constraint in a given situation.

* Op. Cit.

There is no effect on cost associated with values of UTIL less than 0.5, while values of UTIL greater than 0.9 have a very large effect (see Figure A-6).

Values of UTIL for shipborne or mobile applications range from 0.65 to 0.75, airborne applications range from 0.75 to 0.85, and space systems have values close 0.9.

Platform

PLTFM ". . . denotes the operating environment of the software, and is a measure of portability, reliability, structuring, and test and documentation requirements to be provided for acceptable contract performance."*

PLTFM describes the specifications to be satisfied by the software. It represents the degree of testing and documentation associated with it. The lower values denote one-time, in-house software with little or no documentation. Increasing values describe more stringent testing and documentation up to systems such as man-rated space applications. Table A-5 shows typical values of PLTFM.

Complexity

CPLX quantifies the effect on the time required to complete the software development of the organization's readiness to undertake the project.

Values of CPLX range from -0.2 to +0.6. Increasingly negative values describe projects undertaken by experienced groups working on applications very similar to ones that have been done before. The larger positive values would represent projects in which there are one or more factors that have been associated with longer development times. Such factors include inexperienced crews, unfamiliar applications, new hardware or software and so forth. Table A-6 shows the values of CPLX associated with the existence of different personnel and environmental conditions.

* Op. Cit.

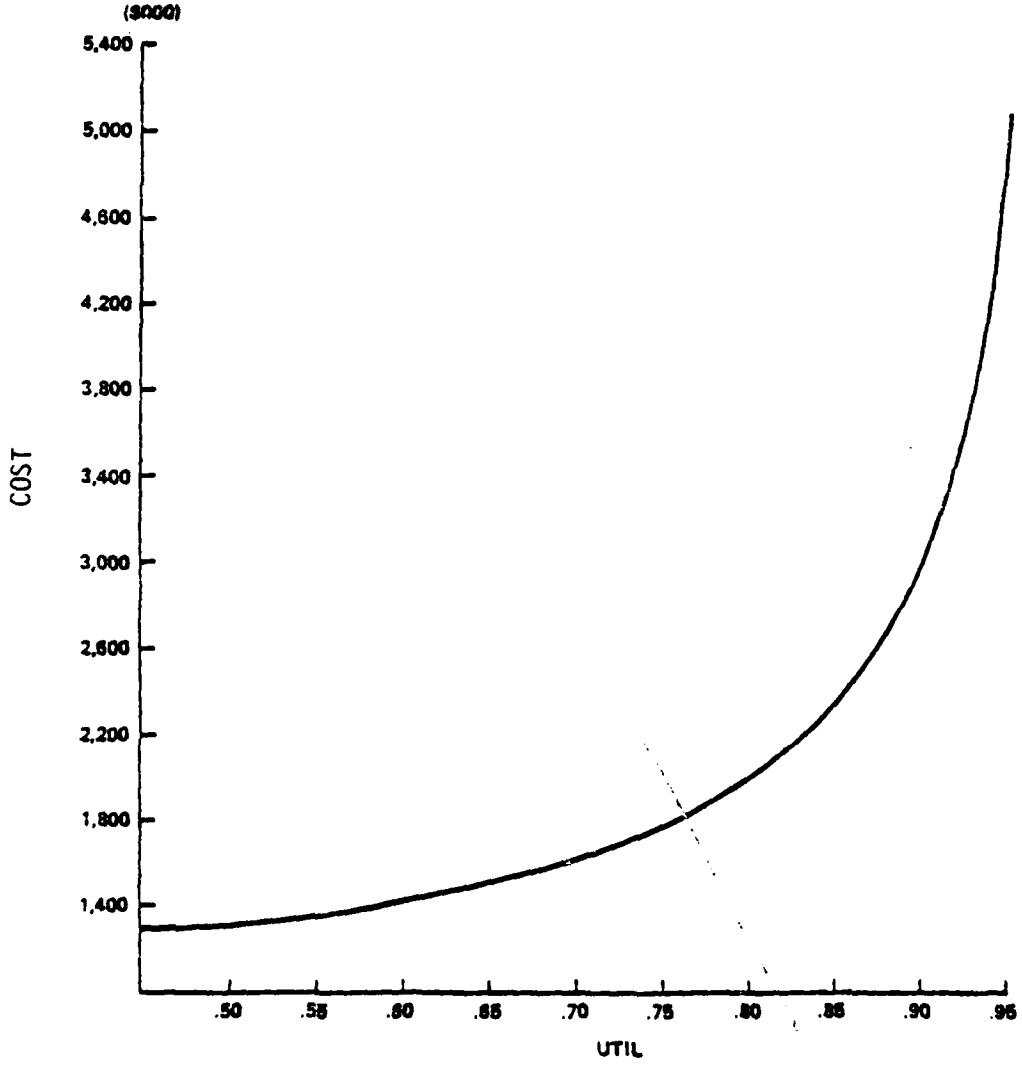


Figure A-6. Effect of UTIL on COST

TABLE A-5

TYPICAL PLTFM VALUES

OPERATING ENVIRONMENT	PLTFM
PRODUCTION CENTER - INTERNALLY DEVELOPED SOFTWARE	0.6-0.8
PRODUCTION CENTER - CONTRACTED SOFTWARE	1.0
MIL-SPEC GROUND	1.2
MILITARY MOBILE (VAN OR SHIPBOARD)	1.4
COMMERCIAL AVIONICS	1.7
MIL-SPEC AVIONICS	1.8
UNMANNED SPACE	2.0
MANNED SPACE	2.5

TABLE A-6

TYPICAL CPLX VALUES

PERSONNEL / ENVIRONMENT	RELATIVELY INEXPERIENCED. MANY NEW HIRES	MIXED EXPERIENCED. SOME NEW HIRES	NORMAL CREW, EXPERIENCED	EXTENSIVE EXPERIENCE. SOME TOP TALENT	EXCEPTIONAL CREW. BEST TALENT IN INDUSTRY
OLD HAT. REDO OF PREVIOUS WORK	1.0	0.9	0.8	0.7	0.6
FAMILIAR TYPE OF PROJECT	1.1	1.0	0.9	0.8	0.7
NORMAL NEW PROJECT	1.2	1.1	1.0	0.9	0.8
NEW HARDWARE OR NEW LANGUAGE	1.3	1.2	1.1	1.0	0.9
NEW HARDWARE AND NEW LANGUAGE	1.4	1.3	1.2	1.1	1.0
HARDWARE DEVELOPED IN PARALLEL	1.5	1.4	1.3	1.2	1.1
MANY CHANGING REQUIREMENTS	1.5	1.4	1.3	1.2	1.1
ADVANCED STATE-OF-THE-ART	1.6	1.5	1.4	1.3	1.2
STATE-OF-THE-ART ADVANCEMENT	1.8	1.7	1.6	1.5	1.4

PRICE S considers CPLX and schedule to be alternative representations of the time required to complete a software development project. If either one is given, the model can compute the other.

Values of CPLX, as in the case of RESO, tend to be constant for a given organization. The calibration mode is used to obtain values from representative past projects. These are used to make estimates for new software development efforts.

Reference Year

PRICE S incorporates a reference calendar that is used to calculate changes in the value of the monetary unit and the rate of technological change.

The reference year for PRICE S cost calculations is 1976. The inflation rate table, RTABLE, is used to adjust calculated costs to the year specified by YEAR.

If no project start date is given, the model assumes a start date of 1 January of YEAR.

YEAR may be used to define the state-of-the art of system development technology. It may be used along with the input TECIMP to represent the decrease in cost associated with expected improvements in development efficiency. TECIMP is included in DTABLE and represents the difference in development technology expected to occur in the interval between YEAR and the start of the project, DSTART.

Cost Multiplier

MULT is a multiplier for all cost values. "Its primary purpose is to adjust all costs to include mark-ups, such as G&A, IR&D, and profit or fee."*

* Op. Cit.

Other Inputs

PRICE S includes many inputs in addition to the required ones described above. These optional inputs serve to define the system in greater detail when desirable and can specify constraints on the project development parameters. They include:

NEWD - The amount of new design required for the software development (Range: 0, 1)

NEWC - The amount of new code required for the software development (Range: 0, 1)

NOTE: NEWD and NEWC are required inputs if APPL is entered rather than calculated from the MIX categories.

SCHEDULE:

DSTART - The date design effort starts

DEND - The date design effort ends

ISTART - The date implementation effort starts

IEND - The date implementation effort ends

TSTART - The date test and integration effort starts

TEND - The date test and integration effort ends

RESOURCE CONSTRAINTS:

DCOST - Average Cost per Man-Month/Hour - Design Phase

DMAX - Maximum Man-Month/Hours per Month - Design Phase

ICOST - Average Cost per Man-Month/Hour - Implementation Phase

IMAX - Maximum Man-Month/Hours per Month - Implementation Phase

TCOST - Average Cost per Man-Month/Hour - Test/Integration Phase

TMAX - Maximum Man-Month/Hours per Month - Test/Integration Phase

NEW DESIGN: The proportion of new design in each mix category required for the software development.

DDAT - Date storage and retrieval

DONL - On-Line communications

DREA - Real-time command and control

DINT - Interactive operations
DMAT - Mathematical applications
DSTR - String manipulation
DOPR - Operating systems

OPTIONAL

DAPP8 - Applies only when MAPP8 and APPL8 are specified

NEW CODE: The proportion of new code in each mix category required for the new development.

CDAT - Data storage and retrieval
CONL - On-Line communications
CREA - Real-time command and control
CINT - Interactive operations
CMAT - Mathematical applications
CSTR - String manipulation
COPR - Operating systems

OPTIONAL

CAPP8 - Applies only when MAPP8 and APPL8 are specified

INTERFACE TYPES:

TDAT - Data storage and retrieval devices
TONL - On-Line communications devices
TREA - Real-time command and control devices
TINT - Interactive devices

INTERFACE QUANTITIES:

QDAT - Number of data storage and retrieval devices
QONL - Number on Of-Line communication devices
QREA - Number of real-time command and control devices
QINT - Number of interactive devices

CAP - Available memory size. UTIL = INST/CAP.

INTEG - Adjustment for system integration cost.

Calibration (ECIRP)

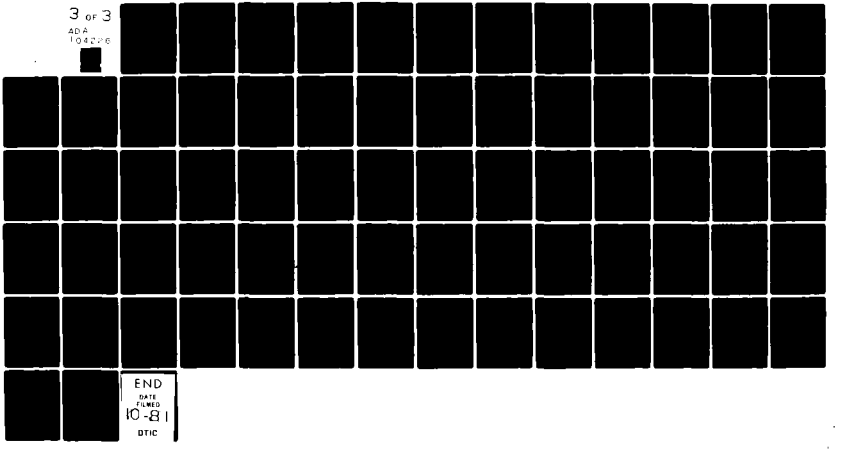
PRICE S incorporates the assertion that many different development projects histories can be associated with software represented by a given set of characteristics. These differences can be attributed to how an organization undertakes a project. PRICE S recognizes two types of project development. An organization may operate in a manner emphasizing tight schedules and higher staffing rates or it may elect to limit staffing and extend the completion time.

The discussion of inputs described the parameter RESO to be associated with the expenditure of project resources and CPLX with the project schedule. ECIRP is an execution mode of PRICE S that uses historical project data to calculate values of these parameters. The values obtained indicate how the organization has historically staffed and scheduled its projects. The prerequisite for obtaining reliable cost estimates with PRICE S is to verify that the values of RESO and CPLX behave in a consistent pattern for the organization. Establishing these values using past projects provides two important parameters that cannot be obtained with any confidence from any outside source.

Data obtained from an organization's records reflects all the definitions and processes peculiar to that organization. These become implicitly represented in the calibration parameters and are reflected in subsequent estimates.

The ECIRP mode is executed by entering a non-zero value for the input TARCST. TARCST is the total development cost for a completed project. Using this value, the project schedule descriptions of system size, application and other characteristics allows the model to calculate values for RESO and CPLX. Repeating the calibration for several projects provides the basis for estimating RESO and CPLX for new projects.

AD-A104 226 GENERAL RESEARCH CORP HUNTSVILLE AL F/G 9/2
AN EVALUATION OF SOFTWARE COST ESTIMATING MODELS.(U)
JUN 81 R THIBODEAU F30602-79-C-0244
UNCLASSIFIED GRC-CR-1-940 RADC-TR-81-144 NL



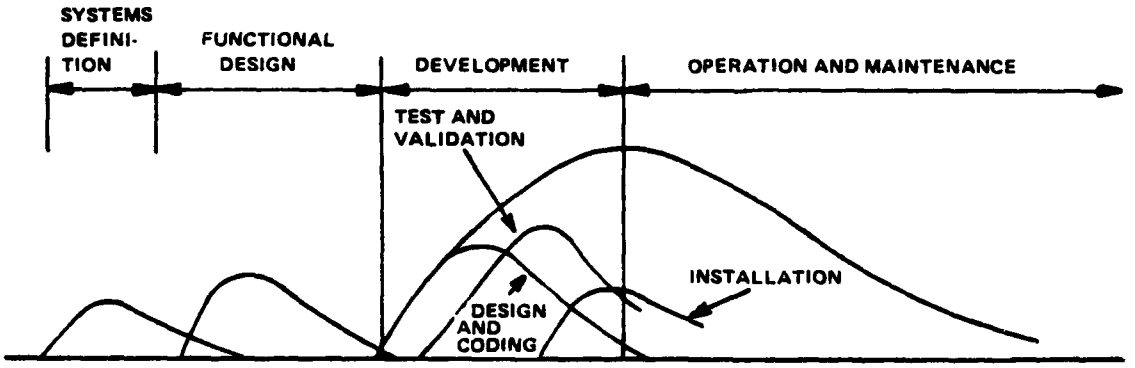
END
DATE FILMED
10-81
DTIC

SLIM

Description of the Model

SLIM (Software Life Cycle Model) is a proprietary software cost estimating model offered by Quantitative Software Management, Inc., McLean, Virginia. The model is presently resident on the American Management Systems time sharing network which provides local dial-up facilities for a variety of low-speed terminals in over 200 cities nation-wide.

SLIM has its origins in the work done by L. H. Putnam at the US Army Computer Systems Command. Putnam applied the hypothesis presented by Norden of IBM* that given linear learning, the rate of expending effort on the solution of problems follows a Rayleigh distribution function over time. Both Norden and Putnam obtained good results by fitting the Rayleigh function to the distribution of effort per unit time over the life cycle of large software development projects. The Rayleigh representation was found to be applicable for both the total life cycle effort and for the component activities such as Design and Coding, and Test and Validation.



Using the Rayleigh distribution has the advantage that the time distribution of effort during the software life cycle is determined by two parameters: the area under the Rayleigh curve, which when applied

* P. V. Norden, Useful Tools for Project Management, Management of Production, M. K. Starr, Ed., Penguin Books, 1970, pp 71-101.

to the rate of expending effort becomes the total life cycle effort; and the time to reach the peak of the effort rate. Putnam showed that for large systems this time is the development time. The result of applying the Rayleigh form to the software development cycle is the equation:

$$\dot{y} = \frac{K}{t_d^2} t \exp\left(\frac{-t^2}{2t_d^2}\right)$$

where:

$\dot{y}$ = the rate of expending effort, e.g., man-months per month;
K = the total life cycle effort, man-months or man-years;
t_d = the development time, months or years;
t = the time from the start of development, months or years.

Putnam then observed that the variable K/t_d^2 was correlated with the subjective difficulty of a system. He calls the ratio "Difficulty" and has determined empirically that for large systems (more than 70000 source statements), the productivity (source statements per man-year of development) is related to the Difficulty by the following equation:

$$\overline{PR} = CnD^{-2/3}$$

where:

$\overline{PR}$ = the average productivity, source statements per man-year of development;
D = Difficulty, K/t_d^2;
Cn = empirical constant.

The portion of the life cycle that produces the code (Design and Coding Phase) defines productivity and this activity constitutes 1/6 the total life cycle effort. The Design and Coding Phase begins at the start of the life cycle and is approximately 95 percent complete at the end of development (some design and coding occurs after system installation). These observed relationships along with the preceding ones define the software equation:

$$S_S = C_K K^{1/3} t_d^{4/3}$$

where:

C_K = the Technology Constant.

Putnam states that the Technology Constant is quantized and it "seems to relate machine throughput (or programmer turn-around, available test time, etc.) and other technology improvements like the chief programmer team, top down structured programming, on-line interactive job submission, etc."*

The final equation developed by Putnam concerns an observed relationahip between the type of development being undertaken, the development time and the difficulty. Putnam shows that the gradient of Difficulty:

$$\nabla D = \frac{D}{t_d} = \frac{K}{t_d^3}$$

is related to the type of development (e.g., stand-alone system, rebuild, composite, etc.). The Difficulty gradient takes on a specific value for each type of system and has the effect of imposing a minimum development time for any system with a given Difficulty and total life cycle effort. Trying to develop the system in a shorter time increases the Difficulty; conversely, increasing the development time decreases the Difficulty.

As a consequence of the Rayleigh/Norden distribution of effort per unit time and the empirical relationships between productivity and D and between ∇D and type of development, Putnam proposes that the following equations govern the life cycle effort for large software systems:

$$S_S = C_K K^{1/3} t_d^{4/3}$$

$$\nabla D = \frac{K}{t_d^3}$$

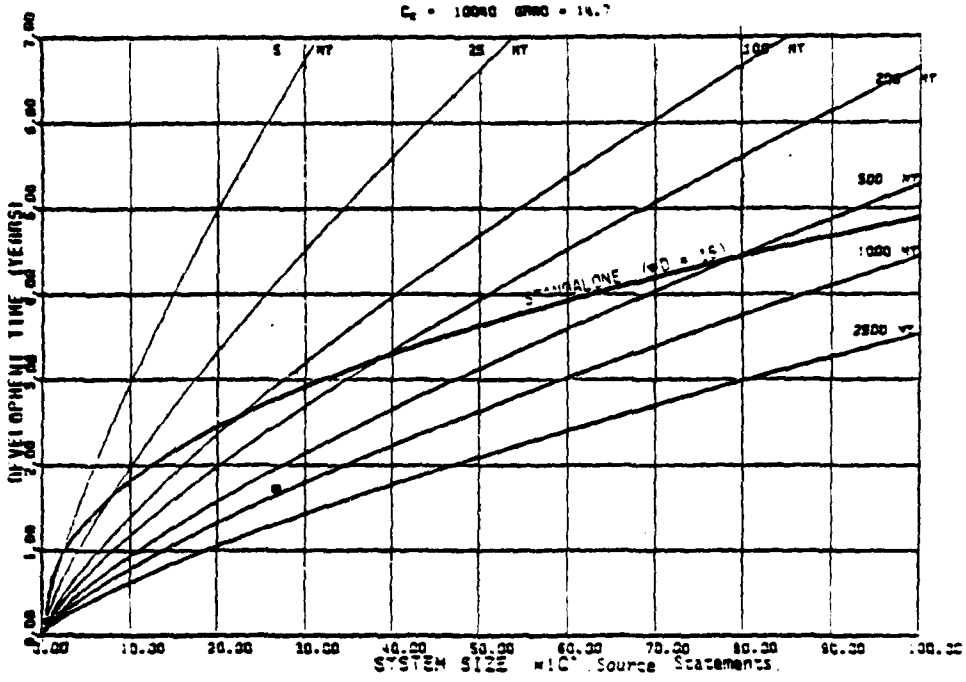
where:

C_K = the Technology Constant and is known for a given environment;

∇D = the Difficulty gradient and is known for a given type of development.

* L. H. Putnam, Measurement Data to Support Sizing, Estimating and Control of the Software Life Cycle, IEEE COMPCON 78, San Francisco, Calif, Mar 1978, p 12.

For a given value of the Technology Constant, and type of development, the software equation and the Difficulty gradient define time-effort-Difficulty trade-offs for any system of a given size.



The method to this point provides a minimum development time and a relationship between effort and development time for a given system and environment. Putnam recommends[1] that the selected development time within the trade-off region be as long as possible within the constraints imposed by the need for the system. As can be seen from the chart above, it is advantageous to extend the development time as much as possible. But in a practical situation there are several additional constraints that affect the development time and effort for a software system. Putnam has incorporated these into a linear programming problem which is one of the operating modes of SLIM. The linear programming solution satisfies the following relationships[2].

---

(1) L. H. Putnam, The Real Economics of Software Development, Quantitative Software Management, Inc., Jun 1980, p 5.

(2) L. H. Putnam, Software Costing and Life Cycle Control, Workshop on Quantitative Software Models, IEEE Cat. No. TH0067-9, Oct 1979, p 29.

$$S_S = C_K K^{1/3} t_d^{4/3}$$ software equation

$$K/t_d \leq \sqrt{e}y_{max}$$ maximum peak manpower

$$K/t_d \geq \sqrt{e}y_{max}$$ minimum peak manpower

$$K/t_d^2 \leq |D|$$ maximum difficulty

$$K/t_d^3 \leq |\nabla D|$$ maximum difficulty gradient

$$t_d \leq$$ contract delivery time

$$\overline{\$/MY}\ (YK) \leq$$ total budgeted amount for development

Providing that a solution exists that satisfies all the constraints, the result is a range of development times and efforts extending from a minimum time solution on one extreme to a minimum effort or cost on the other.

	TIME	MAN-MONTHS	COST (X $1000)
MINIMUM TIME	21.93	399	1662
	22.43	364	1519
	22.93	334	1390
	23.43	306	1276
MINIMUM COST	24.00	278	1159

The notion that the cost of software development should be dependent on management strategy as an explicit input to the cost estimating model is unique. Although other models may be used to the same end, SLIM includes it as part of the model structure.

Another important feature of SLIM is the representation of the uncertainty in the primary estimating variable and its effect on the development time and effort. Program size can be represented as a range of possible values for system-level estimates or as smallest, largest, and most likely values for component estimates. In either case, an expected value and variance is established for each size measure and

these are used to perform a Monte Carlo type simulation of the minimum development time and its related effort. The result is statistical distributions of times and efforts that can be used to establish time and effort values associated with different values of risk. The model uses these relationships in several of its operating modes. The applications of these relationships are described in the discussion of the model outputs.

Having established the basic life cycle cost estimates for a given development situation, the model calculates "front end" effort and time as a function of the development effort. It also calculates computer resource requirements and documentation size.

The description of SLIM presented here is based on materials that have been published widely by Putnam and that are derived in large part from his work at the Army Computer Systems Command. The methods are applicable to systems larger than 70000 source statements. The extension of the model to systems between 10000 and 70000 statements is held to be proprietary and there is no available description of the method used to make estimates for systems in this range of the size variable. The evaluated model rejects systems with fewer than 10000 statements. A new version of SLIM is now available that accepts systems as small as 5000 statements.

Outputs

Development Time

The development cycle begins with the detailed design of the system programs. It is assumed that the system requirements and the system specifications are completed in activities that precede the development cycle. Development ends when the system completes its acceptance testing and is released to the user. The Development Time is the elapsed time in months extending from the start of program design until the system is released.

Development Effort

All personnel hours, both direct and indirect, expended during the period defined by the Development Time are included in the development

effort. In many models the precise definition of the indirect hours is an important factor that affects the accuracy of the predictions. The effects of variations in the definitions are minimized in SLIM (as in any models that are self-calibrating) because the important model parameters are calculated from historical data that should reflect the same cost accounting practices as will be applied to the project being estimated. In using such models it is important that the historical data be representative of the estimated project in such definitions as types of personnel hours to be recorded and reported against the project, the level of management reported directly and indirectly and the method of distributing the overhead charges.

The indirect effort is incorporated into the SLIM estimates by way of the Technology Factor. The Technology Factor is obtained using the calibration mode of the model from data describing past projects. The Technology Factor includes a factor related to the organizational productivity. The average productivity is defined as the ratio of the total end product code to the total effort to produce the code. The total effort includes all direct and indirect activities.* Therefore, as long as the size of the system from historical records and the total effort to produce the system are measured on the same basis as the project being estimated, the Technology Factor obtained from the historical data will be appropriate and the details of the definitions of total effort need not be specified as part of the model definitions.

Development Cost, Constant Dollars

The Development Cost is obtained from the Development Effort using a constant cost per unit effort (labor rate) measured in dollars per man year for a specified year. Care must be exercised to ensure that the indirect and overhead costs are consistent with the definitions used to obtain the Technology Factor. If they are not, omissions or double-counting will occur.

---

* L. H. Putnam, A General Empirical Solution to the Macro Software Sizing and Estimating Problem, IEEE Transactions on Software Engineering, Vol. SE-4, No. 4, Jul 1978. p 353.

Development Cost, Inflated Dollars

Costs are expressed in current year dollars using an inflation rate specified in the model inputs.

MODEL OUTPUT OPTIONS

Simulation Option

This is the primary operative mode of SLIM. It performs the calculations that are reflected in almost all the other options. The mean and standard deviation are estimated for the following:

- System Size,
- Minimum Development Time,
- Development Effort,
- Development Cost (inflated and uninflated dollars).

A sensitivity profile is presented that shows the change in Development Time, Effort and Cost with System Size.

A consistency check is made with similar systems as represented by the RADC data base. Calculated values of development effort, time, and productivity and average number of personnel are compared with the 90 percent range of values taken from the data base for the given system size. The model indicates whether the estimated values are within the range, above it, or below it.

The variance in the estimates comes from the assumed uncertainty in the size estimate, the cost per unit effort, and the difficulty gradient.

Linear Program Option

The user specifies the constraints operating on the system development project and the Linear Program Option calculates various time and effort alternatives that satisfy the constraints. The constraints include:

- Maximum Development Cost,
- Maximum Development Time,
- Maximum Number of People at Peak Staffing,
- Minimum Number of People at Peak Staffing.

The software equation is satisfied subject to the above constraints. The model indicates if no solution exists. If a solution is possible, SLIM prints a matrix showing the time, effort, and cost associated with a minimum cost project and a minimum time project. Since these two solutions represent only the extremes of the feasible region, the model also presents effort and cost for a number of intermediate development times.

Front-End Option

Low, expected, and high values of time and effort are calculated for the activities that precede the Development Phase of the software life cycle. These include:

- Feasibility Study,,
- Functional Design.

The estimates are said to be made using IBM data* and the values estimated for the Development Cycle.

Life Cycle Option

SLIM calculates monthly, quarterly, or yearly estimates of the number of people, the cost, and the cumulative cost for the system life cycle. The mean and standard deviation is presented for each value.

Risk Analysis Option

Tables are presented that indicate the probability that time, effort, and cost to develop the system will not exceed the given amounts. From these outputs it is possible to make an inference such as: "There is a 95 percent probability given the input assumptions that the system development will not take longer than 22.3 months." Similar statements can be made for effort and cost.

---

* SLIM User's Guide, Quantitative Software Management, Inc.

Benefit Analysis Option

Using the user-specified economic life of the system and the desired annual rate of return, the model calculates the discounted cash flow value of the system that amortizes its development and maintenance cost.

New Schedule Option

The Simulation option provides estimates of time, cost, and effort based on the minimum time to develop a given system. The New Schedule option permits the user to specify times greater than the minimum development time and obtain corresponding estimates of effort and cost. The results are compared with similar size system experience as in the case for the Simulation option. The Manloading, Cashflow and Life Cycle options may be executed and reflect the new schedule.

Design to Cost Option

Given an effort less than that established by the minimum time (maximum effort) solution will result in a new development time and cost. Consistency checks and subsequent executions of the Manloading, Cashflow, and Life Cycle options are executed in the same manner as the New Schedule option.

Desigr to Risk Option

The user is asked to choose among three levels of risk (.99, .95, and .90) describing the probability of exceeding a user-specified maximum development time. The model calculates expected values and standard deviations of development time, effort, and cost associated with the selected level of risk. The Manloading, Cashflow, and Life Cycle options may be executed using the new parameters.

Other Options

SLIM has several additional output options including:

Manloading by Project Month
Cashflow by Project Month
Major Milestones
Monthly Code Production
Monthly Computer Hours
Documentation
PERT Sizing

Inputs

SLIM has three primary inputs: System Size in number of developed source statements, Level or Difficulty Gradient, and Technology Factor. The System Size is estimated by someone familiar with the functions to be automated and is the primary descriptor of the work to be done. The other two inputs affect the type of effort involved and the development environment. These are obtained from user experience with previous SLIM estimates and by calibrating the model using historical data that is representative of the project to be estimated. The process of obtaining these values is described below in the section on calibration.

Additional input parameters describe other properties of the system and the development process.

System Size

The system size is described in terms of the number of executable source language statements to be written. Data declaration and input/output statements are included in the size measure, but comment statements are not. SLIM accepts two alternate methods for describing system size. The first is suggested for use during the early phases of the life cycle before the system functions are defined. The user makes an estimate of the the possible range of values for the total system size. The model uses this range to calculate the expected value of the system size and its standard deviation.

The representation of the system size as a random variable is used to calculate the effect of the uncertainty in the size measure on the model estimates. Using the Monte Carlo technique, SLIM performs repeated calculations of the output values using values of the system size defined by a normal distribution with the mean and standard deviation calculated as described above. The statistics of the outputs (mean and standard deviation) are printed and these values are used in executing several of the model options. Ultimately this approach gives the user an explicit statement of the risk assiciated with the model predictions.

The second method used by SLIM to calculate the system size requires the user to make estimates of the sizes of each of the system functions. It is suggested that several analysts make three estimates of the size of each system function: the least possible number of statements, the greatest, and the most likely. These are used to make composite estimates for each function by combining the range values and calculating the mean of the expected values for each function. The three values obtained for each function are used to calculate the system expected value and standard deviation using the relationships:

$$E(S_s) = \frac{1}{N} \sum_{i=1}^{N} E_i$$

$$E_i = \frac{a_i + 4m_i + b_i}{6}$$

$$\hat{\sigma}_{S_s} = [\sum_{i=1}^{N} \hat{\sigma}_i^2]^{\frac{1}{2}}$$

$$\hat{\sigma}_i = \frac{b_i - a_i}{6}$$

where:

$E(S_s)$ = the expected system size in number of source statements

N = the number of system functions

E_i = the expected value of the size of the ith function

a_i, b_i = the range of the size of the ith function from smallest to largest

m_i = the most likely value of the size of the ith function

$\hat{\sigma}_{S_s}$ = the estimated standard deviation of the system size

$\hat{\sigma}_i$ = the estimated standard deviation of the size of the ith function.

The values of the system expected size and its standard deviation are used in the same way as the first method described above.

Level

This parameter is related to the Difficulty Gradient, K/t_d^3, discussed above. The Difficulty Gradient was observed by Putnam to assume discrete values that are representative of the type of development associated with the system being estimated. Consequently, Level measures, " . . . the amount of interfacing, new design, and concurrent programming that will go on during development".* Level takes on integer values from 1 to 5 depending on the following considerations:

(1) The system is entirely new - designed and coded from scratch. It has many interfaces and must interact with other systems within a total management information system structure.

(2) This is a new stand-alone system. It is also designed and coded from scratch but is simpler because the interface problem with other systems is eliminated.

* SLIM User's Guide, p 4-12.

(3) This is a rebuilt system where large segments of existing logic exist. The primary tasks are recoding, integration, interfacing, and minor enhancements.

(4) This is a composite system made up of a set of independent subsystems with few interactions and interfaces among them. Development of the independent subsystems will occur with considerable overlap.

(5) This is a composite system made up of a set of independent subsystems with a minimum of interactions and interfaces among them. Development of the independent subsystems will occur virtually in parallel.

Past data have shown that large systems (>200,000 lines) are typically of Type 3, 4, or 5.

Technology Factor

The Technology Factor is an integer parameter related to the Technology Constant described above. SLIM accepts values in the range from 0 to 22. The 0 value, however, is simply a code that indicates that the model is to provide the Technology Factor. For large systems the Technology Constant can be expressed as:

$$C_k = \frac{2.49}{6} C_n$$

where:

C_k = The Technology Constant

2.49 = A constant that represents the ratio of total life cycle effort to the design and coding effort - it represents the overhead labor associated with code production. It is valid for large systems, but becomes a variable for systems in the range of 18000 to 70000 statements.

6 = A constant that indicates that 1/6 the life cycle effort is expended on logic design and code production. Valid for large systems; variable for intermediate size systems.

C_n = An empirical constant that was shown above to relate coding productivity and system difficulty. It assumes discrete values.

For large systems, then, the Technology Constant can be seen to account for the rate of code production for a given system Difficulty, overhead labor and the distribution of the life cycle effort. The Technology Factor, which is used to index the Technology Constant and which preserves its discrete property, is said to be " . . . a measure of the state of technology of the human-machine system"*, environmental influences and functional complexity of the system.

The interpretation of the Technology Factor is the same for smaller systems as it is described above for large systems, but the relationship shown is modified in a way that is proprietary.

The Technology Factor is obtained by calibrating the model using historical data that are representative of the project to be estimated. The factor should be stable in a given organization, but should be expected to change to reflect differences in:

- Computer access and availability
- Software support tools, language
- Use of modern programming practices
- Type of application
- Staff experience
- Customer relationship

The SLIM manual indicates that few organizations are represented by a Technology Factor greater than 14 or less than 5.

---

* L. H. Putnam, A General Empirical Solution, etc., op. cit. p 353.

Additional Inputs

SLIM requires a number of inputs in addition to the ones described above.

MONTH, YEAR = the month and year when detailed design of the system will start. MONTH is an integer between 1 and 12; YEAR is an integer between 40 and 90.

LABOR RATE - the fully burdened average $/MY at the user's organization.

STDDEV - the uncertainty associated with the above LABOR RATE.

INFLATION RATE - the anticipated inflation rate at project start

ONLINE - the proportion of development that will occur in online, interactive mode.

DEVELOPMENT TIME - the proportion of the development computer that is dedicated to this development effort.

PRODUCTION TIME - the proportion of the available capacity of the development computer that is used for other production work.

HOL - the proportion of the system that will be coded in a higher order language.

LANGUAGE - the primary language to be used; should correspond to the legend below.

(1)	APL	(4)	FORTRAN	(7)	ALGOL	(10)	ASSEMBLER
(2)	PL/I	(5)	BASIC	(8)	JOVIAL	(11)	RPG
(3)	COBOL	(6)	CMS	(9)	PASCAL-ADA	(12)	OTHER

UTILIZATION - the proportion of the memory of the target machine that will be utilized by the end system.

REAL TIME CODE - the proportion of code which is devoted to real time or time critical functions.

MODERN PROGRAMMING PRACTICES - the 4 variables include: STRUCTURED PROGRAMMING, DESIGN/CODE INSP, TOP-DOWN DEVELOPMENT, and Chief Programmer Team usage. The responses for each of these variables should correspond to the legend below.

(1) <25% (2) 25-75% (3) >75%

TYPE - description of the type of software system:

(1) Real time or time critical system
(2) Operating system
(3) Command & control
(4) Business application
(5) Telecommunication & message switching
(6) Scientific system
(7) Process control

PERSONNEL EXPERIENCE - the 4 variables include: OVERALL, SYSTEM TYPE, LANGUAGE, and HARDWARE. The responses are used to get an indication of the level of personnel experience -- overall, on a system of similar size and application, with the programming language to be used on this effort, and on the development machine.

(1) MINIMAL (2) AVERAGE (3) EXTENSIVE

The Technology Factor describes the development environment. As was described above, the Technology Factor is an indicator of the efficiency with which effort (and therefore cost) is expended to obtain the desired software system. Values of the Technology Factor can be obtained from data describing completed projects by using the SLIM calibration option. Given the values of:

- Size
- Development Effort
- Development Time

for one or more projects, the model calculates the Technology Factors that would have produced the indicated experience. The user is cautioned to examine any Technology Factor outside the "reasonable range." This is indicated by an asterisk in the printout.

The Technology Factor may vary for selected projects taken from a single organization. This may be caused by differences among the projects in computer access, software support, management methods staff experience, language, user characteristics, requirements stability and functional complexity. It is the user's responsibility to examine these possibilities and to verify that the project being estimated is compatible with the selected value.

Reference

SLIM User's Guide, Quantitative Software Management, Inc.

TECOLOTE

Description of the Model

The Tecolote provisional software cost estimating model was developed to predict cost and resources needed to develop tactical software. Specifically it was derived using data representing Navy fire control systems designed to operate against air and sea threats. These two classes of fire control systems present different software requirements because of the effects of threat speed on system response speed.

The justification for the model form is the hypothesis that development effort is determined by software size, system time criticality and system fast storage capacity. The model is limited to tactical systems characterized by time criticality, that is, where the time required to access the fast storage memory is comparable to the speed with which the computer is capable of moving data during processing.

The software resource-driving factors (storage capacity and time criticality) were assumed to be predictable in terms of the related threat characteristics. Data from five Navy software developments were used to relate the storage and time requirements to threat size for two speed regions representing the air and sea threats.

A relationship was derived between "delivered code" (the total code developed including drivers and simulators) and "operational code" (the code that ends up in the operational computer). All code size measurements are in terms of machine instructions. The number of machine instructi is taken to be the same as the number of words of computer storage required to store the program.

Operating instructions are related to total fast storage capacity using one sea threat data point and two air threat points. A relationship between total delivered instructions and operational instructions is obtained from two data points.

Man-months of direct labor are shown as functions of first total operating instructions and then total delivered instructions.

A matrix of the above relationships was prepared (Table A-7). Its purpose is ". . . for evaluating software proposals from the standpoint of software design as well as software costs." It should be noted that there are redundancies in the relationships and that the model does not indicate which relationship should be preferred in any given estimating situation.

Labor and computer costs were presented in 1973 dollars as functions of direct labor man-months. These were obtained from the only project for which costs were available.

Reference

Brad C. Frederick, A Provisional Model for Estimating Computer Program Development Costs, Tecolote Research, Inc., TM-7, Dec. 1974.

TABLE A-7

SUMMARY OF PROVISIONAL SOFTWARE ESTIMATING RELATIONSHIPS (SEE NOTE A)

INPUTS / OUTPUTS	M, TOTAL MAN-MONTHS LABOR	D, TOTAL DELIVERED INSTRUCTIONS (THOUSANDS)	O, TOTAL OPERATING INSTRUCTIONS (THOUSANDS)	AIR THREATS (B)		SEA THREATS	
				S, TOTAL WORDS FAST STORAGE (THOUSANDS)	T, TARGETS TERMINAL-TRACKED	S, TOTAL WORDS FAST STORAGE (THOUSANDS)	T, TARGETS TERMINAL-TRACKED
TOTAL DEVELOPMENT COST FY 73 $M	$0.0043(M)$	$0.01(D)^{1.18}$	$0.01(O)^{1.24}$	$0.0026(S)^{1.79}$	$0.30(T)^{1.88}$	$0.0043(S)^{1.79}$	$0.19(T)^{1.88}$
TOTAL MAN-MONTHS LABOR		$2.43(D)^{1.18}$	$2.52(O)^{1.24}$	$0.59(S)^{1.79}$	$69(T)^{1.88}$	$1.01(S)^{1.79}$	$45(T)^{1.88}$
TOTAL DELIVERED INSTRUCTIONS (THOUSANDS)			$1.03(O)^{1.05}$	$0.30(S)^{1.51}$	$17(T)^{1.59}$	$0.48(S)^{1.51}$	$12(T)^{1.59}$
TOTAL OPERATING INSTRUCTIONS (THOUSANDS)				$0.31(S)^{1.44}$	$14(T)^{1.51}$	$0.48(S)^{1.44}$	$10(T)^{1.51}$
TOTAL WORDS FAST STORAGE (THOUSANDS)					$14.30(T)^{1.05}$		$8.30(T)^{1.05}$

NOTES: (A) COSTS ASSUME $3,930/MM FOR LABOR, $77/COMPUTER HOUR, AND 4.23 COMPUTER HOURS/MM.

(B) USE AIR THREAT COLUMN IF MAXIMIM THREAT SPEED IS IN THE 250-700 M/SEC RANGE.

Wolverton

Description of the Model

Estimates of routine size are converted to costs using cost per instruction values that are functions of the routine tvpe and complexity. The costs are fully burdened and when summed for all the system routines represent the total system development cost. Development extends from analysis and design through operational demonstration. A matrix of ratios is used to allocate the total cost to 7 phases with each phase divided into up to 25 activities. This allocation is compared from the standpoints of staff, schedule, and general credibility.

The model, then, is a combination of formal algorithm and judgement. It has been used successfully at TRW. As described by Wolverton, it features a data base of historical data that provide the necessary cost per instruction and allocation values. The procedure is adaptable to any new environment by creating a new data set representing local definitions of phases and activities and burdened cost conventions. In fact, Wolverton cautions that the given values of cost per instruction are for illustration and users should prepare their own values.

TRW has computerized the maintenance of the cost data base and the allocation process. Given the inputs of size and complexity, the system calculates the cost allocations and facilitates any subsequent adjustments. Since most models are used in a similar manner, even if the procedure for using the model does not say so, there should be no compromise of the model's performance if the evaluation is based on a single estimate of costs. Other adjustments that are necessary to execute the model in different environments will be discussed later.

The estimating procedure begins by identifying all the routines comprising the system. Each routine size, category, and relative degree of difficulty are estimated by knowledgeable persons.

The categories that have "stood the test of usage" at TRW are:

- Control routine
- Input/Output routine
- Pre or Post algorithm processor
- Algorithm
- Data Management routine
- Time-Critical processor

Relative difficulty is indicated by six levels depending on whether a routine is Old or New and then by simply: Easy, Medium or Hard.

The cost per instruction for the 36 different attributes (6 software categories by 6 levels of difficulty) is given in Figure A-7. Multiplying the cost per instruction for each routine by its number of object instructions and summing the products for all the routines yields the estimated total development cost.

The development cost is allocated to the following 7 phases using proportions for each phase that were obtained from the historical data base.

A. Performance and Design Requirements
B. Implementation Concept and Test Plan
C. Interface and Data Requirements Specification
D. Detailed Design Specification
E. Coding and Auditing
F. System Validation Testing
G. Certification and Acceptance Demonstration

Then, the cost for each phase is divided into up to 25 activities (Tables A-8 and A-9).

A matrix of computer hours by phase and software type is used to estimate computer usage costs for development.

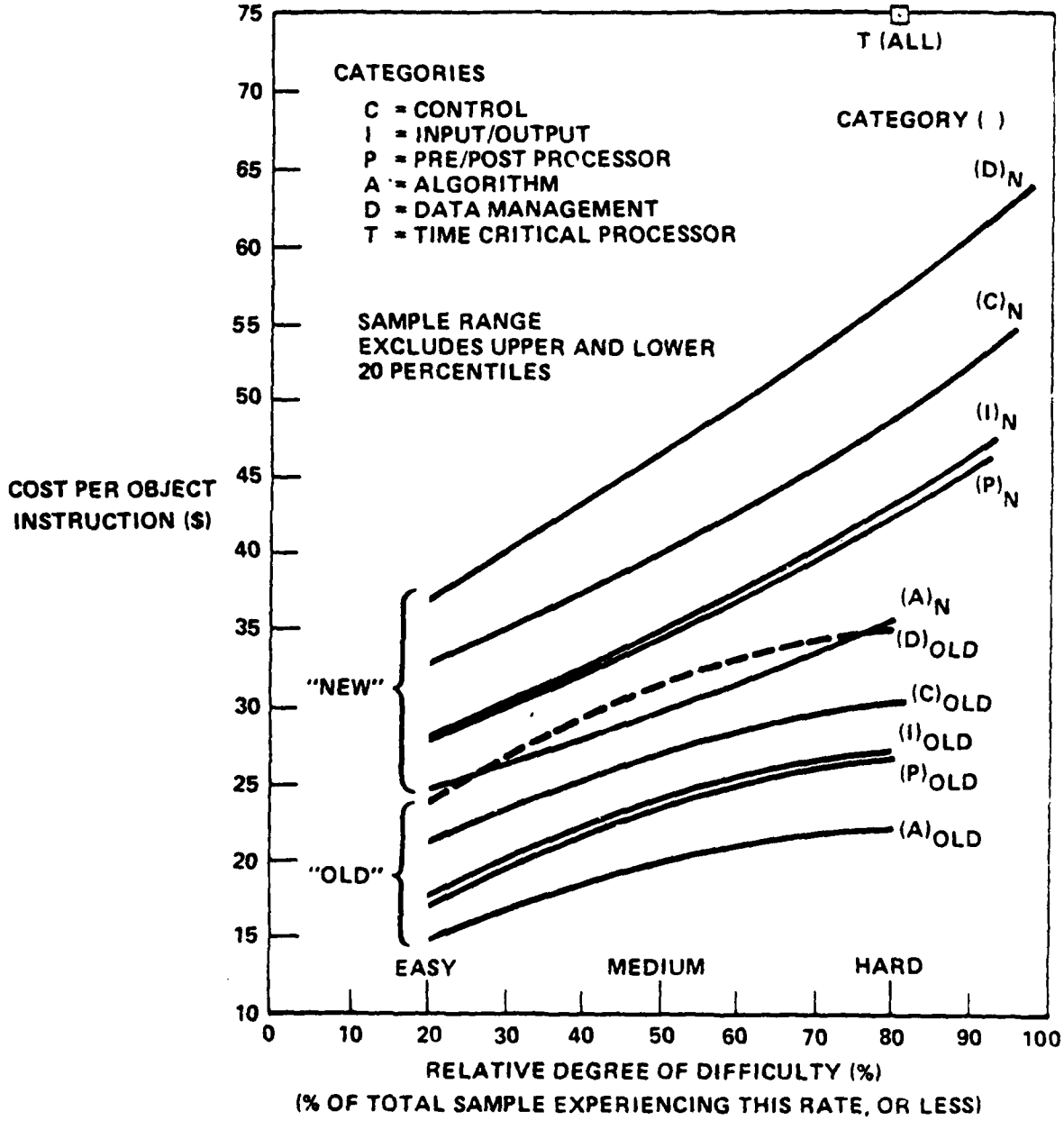


Figure A-7. Cost Per Object Instruction Versus Relative Degree of Difficulty

TABLE A-8

ACTIVITIES AS A FUNCTION OF SOFTWARE DEVELOPMENT PHASE

ACTIVITY \ DEVELOPMENT PHASE		PHASE A	PHASE B	PHASE C	PHASE D	PHASE E	PHASE F	PHASE G	PHASE H
		PERFORMANCE AND DESIGN REQUIREMENTS	IMPLEMENTATION CONCEPT AND TEST PLAN	INTERFACE AND DATA REQUIREMENTS SPECIFICATION	DETAILED DESIGN SPECIFICATION	CODING AND AUDITING	SYSTEM TESTING	CERTIFICATION AND ACCEPTANCE	OPERATIONS AND MAINTENANCE
MANAGEMENT	1	PROGRAM MANAGEMENT	PROGRAM MANAGEMENT	PROGRAM MANAGEMENT	PROGRAM MANAGEMENT	PROGRAM MANAGEMENT	PROGRAM MANAGEMENT	PROGRAM MANAGEMENT	PROGRAM MANAGEMENT
	2	PROGRAM CONTROL	PROGRAM CONTROL	PROGRAM CONTROL	PROGRAM CONTROL	PROGRAM CONTROL	PROGRAM CONTROL	PROGRAM CONTROL	PROGRAM CONTROL
REVIEWS	3		PRELIMINARY DESIGN REVIEW (PDR)	INTERFACE DESIGN REVIEW (IDR)	CRITICAL DESIGN REVIEW (CDR)	SYSTEM TEST REVIEW (STR)		ACCEPTANCE TEST REVIEW (ATR)	OPERATIONAL CRITIQUES
DOCUMENTS	4	DOCUMENT AND EDIT	DOCUMENT AND EDIT	DOCUMENT AND EDIT	DOCUMENT AND EDIT	DOCUMENT AND EDIT		DOCUMENT AND EDIT	DOCUMENT AND EDIT
	5	REPRODUCTION	REPRODUCTION	REPRODUCTION	REPRODUCTION	REPRODUCTION		REPRODUCTION	REPRODUCTION
REQUIREMENTS AND SPECIFICATIONS	6	REQUIREMENTS DEFINITION	PDR MATERIAL	EVENT GENERATION INTERFACE	PRODUCT CONFIG DETAILED TECH DESCRIPTION (PART II) (WITHOUT LISTINGS)	TECHNICAL DESCRIPTION UPDATE	PRODUCT CONFIG DETAILED TECH DESCRIPT UPDATE (PART II) (WITH LISTINGS)	REQUIREMENTS CERTIFICATION	SOFTWARE PROBLEM REPORTS (SPR)
	7	REQUIREMENTS ALLOCATION	PERFORMANCE AND DESIGN REQUIREMENTS (PART I)	COMMAND DEFINITION I/F		TRAINING DOCUMENTATION		FINAL DOCUMENTATION UPDATE (PART II)	
	8			TELEMETRY DEFINITION I/F			INTERFACE SPECIFICATIONS UPDATE		
	9			OPERATIONAL ENVIRONMENT I/F					
DESIGN	10	TRADE STUDIES	TRADE STUDIES	TRADE STUDIES	TRADE STUDIES				
	11	INTERFACE REQUIREMENTS	FUNCTIONAL DEFINITION	DATA DEFINITIONS	ALGORITHM DESIGN	ALGORITHM UPDATE	PROGRAM UPDATE		
	12	HUMAN INTERACTION	STORAGE AND TIMING ALLOCATION		PROGRAM DESIGN				
	13	STANDARDS AND CONVENTIONS	DATA BASE DEFINITION	DATA BASE DESIGN		DATA BASE UPDATE	DATA BASE UPDATE		
	14		SOFTWARE OVERVIEW (PRELIMINARY)		SOFTWARE OVERVIEW UPDATE				
CODING	15				PROTOTYPE CODING	OPERATIONAL CODING			
TESTING, CONFIGURATION CONTROL AND QAA	16		PRODUCT AND CONFIGURATION CONTROL	PRODUCT AND CONFIGURATION CONTROL	PRODUCT AND CONFIGURATION CONTROL	PRODUCT AND CONFIGURATION CONTROL	PRODUCT AND CONFIGURATION CONTROL	PRODUCT AND CONFIGURATION CONTROL	PRODUCT AND CONFIGURATION CONTROL
	17		DATA BASE CONTROL	DATA BASE CONTROL	DATA BASE CONTROL	DATA BASE CONTROL	DATA BASE CONTROL	DATA BASE CONTROL	DATA BASE CONTROL
	18	TEST REQUIREMENTS	TEST PLANS	INTERFACES	TEST PROCEDURES	DEVELOPMENT TESTING PLANNING	SYSTEM TEST PLANNING	ACCEPTANCE AND TEST PLANNING	INTEGRATION TESTING
	19					DEVELOPMENT TEST	SOFTWARE SYSTEM TEST	ACCEPTANCE DEMONSTRATION	SPR CLOSURE
	20						HARDWARE/ SOFTWARE SYSTEM TESTING		
	21	ACCEPTANCE TEST REQUIREMENTS	QUALITY AND RELIABILITY ASSURANCE PLANS	QA AND RA MONITORING	QA AND RA MONITORING	QA AND RA MONITORING	QA AND RA MONITORING	QA AND RA MONITORING	QA AND RA MONITORING
	22					TEST SUPPORT	TEST SUPPORT	TEST SUPPORT	TEST SUPPORT
	23								
OPERATIONS	24		OPERATIONAL CONCEPT	OPERATIONAL TIMELINE	OPERATIONAL CONCEPT UPDATE	USER'S MANUAL (PRELIMINARY)	OPERATIONAL TIMELINE UPDATE	USER'S MANUAL UPDATE	INTEGRATION SUPPORT
	25		TRAINING PLAN		TRAINING PLAN UPDATE	TRAINING	TRAINING	TRAINING AND REHEARSAL	TRAINING AND REHEARSAL

TABLE A-9

COST MATRIX DATA, SHOWING ALLOCATION OF RESOURCES AS A FUNCTION OF ACTIVITY BY PHASE

ACTIVITY	PHASE A (8)	B (19)	C (3)	D (14)	E (23)	F (21)	G (12)	H (0)
1	10	6	8	6	7	5	10	2
2	8	3	3	3	3	3	3	5
3		6	4	6		3	8	5
4	13	8	5	6	5	5	4	2
5	5	2	2	3	3	2	2	1
6	22	8	7	12	3	7	5	8
7	10	8	7		2		6	
8			7			5		
9			6					
10	17	10	10	8				
11	2	10	10	9	7	6		
12	2	5		13				
13	4	7	10		3	5		
14		4		3				
15				5	25			
16		4	4	5	4	4	10	10
17		3	6	5	6	5	8	10
18	5	6	3	8	5	2	5	10
19					10	15	14	5
20						10		
21	2	4	5	5	7	9	9	3
22					5	6	8	4
23								
24		4	3	2	3	5	3	25
25		2		1	2	3	5	10

Outputs

Development Cost

The given cost values are in 1972 dollars. The value of cost results from applying "bid rates" to labor costs which accounts for fringe benefits, overhead, administrative expenses and other indirect costs. Documentation and travel costs are added to the labor costs. Finally, estimates are made of the computer costs. The distribution of the costs by phases and activities were described above.

Development Effort

Cost is not a suitable basis for evaluating the different software estimating models because of differences in accounting practices among organizations and because of inflation. Therefore, the Wolverton cost values were converted to manmonths using an average burdened cost per manmonth of $4600. This value was obtained from the article describing the TRW estimating procedure and, therefore, should be representative of the cost environment.

Inputs

Object Instructions

The model input measure of size is applied to programs or routines. These are taken to be functionally distinct elements of a system that would be developed independently then intergrated into the delivered system. It is expected that these would be independently operable using test drivers. Such a definition is consistent with industry usage. The reference document is not specific on this point. The term "instructions" is taken literally. This means estimating the number of instructions in the executable program exclusive of any data areas. The number of instructions may be estimated by obtaining the words of memory occupied by the executable code and dividing by the average words per instruction.

Software Categories

Each routine is characterized according to one of the following categories:

C. Control Routine. Controls execution flow and is nontime critical.

I. Input/Output Routine. Transfers data into and out of computer.

P. Pre-or Post Algorithm Processor. Manipulates data for subsequent processing or output.

A. Algorithm. Performs logical or mathematical operations.

D. Data Management Routine. Manages data transfer within the computer.

T. Time Critical Processor. Highly optimized machine-dependent code.

Degree of Difficulty

Wolverton indicates that any numeric representation of complexity may be used. The main purpose is to distribute the cost per instruction values over the range of experience for a given category of software. He suggests a simple designation of old or new, depending on a loose interpretation of the amount of reusable code, and easy medium or hard compared with other programs in the same category.

Reference

L. H. Putnam, R. W. Wolverton, Tutorial, Quantitative Management: Software Cost Estimating, IEEE Computer Society, No. EHO 129-7, Nov 1977.

APPENDIX B

WORK BREAKDOWN STRUCTURE

LEVEL 1.

1. DEFINITION
2. CODING
3. DATA CONVERSION
4. INFORMAL TEST AND INTEGRATION
5. FORMAL TEST AND INTE RATION
6. INSTALLATION
7. DEVELOPMENT FACILITIES
8. TRAINING
9. MANAGEMENT

LEVEL 2.

1. DEFINITION
    - .1 SYSTEM LEVEL
    - .2 SYSTEM SEGMENT LEVEL
    - .3 CPCI LEVEL
2. CODING
    - .1 CPCI LEVEL
    - .2 CPRC LEVEL
3. DATA CONVERSION
    - .1 CPCI LEVEL
4. INFORMAL TEST AND INTEGRATION
    - .1 CPCI LEVEL
    - .2 CPRC LEVEL
5. FORMAL TEST AND INTEGRATION
    - .1 SYSTEM LEVEL
    - .2 SYSTEM SEGMENT LEVEL
    - .3 CPCI LEVEL
6. INSTALLATION
    - .1 SYSTEM LEVEL
7. DEVELOPMENT FACILITIES
    - .1 SYSTEM LEVEL
    - .2 SYSTEM SEGMENT LEVEL
8. TRAINING
    - .1 SYSTEM LEVEL
    - .2 CPCI LEVEL
9. MANAGEMENT
    - .1 SYSTEM LEVEL
    - .2 SYSTEM SEGMENT LEVEL
    - .3 CPCI LEVEL

LEVEL 3.

1. DEFINITION

.1 SYSTEM LEVEL

.1 System Requirements Collection and Definition
.2 System Requirements Evaluation
.3 System Design
.4 System Design Verification
.5 System Change Proposal Evaluation and ECP Preparation
.6 System Requirements Documentation
.7 System Design Documentation
.8 System Definition Monitoring and Direction

.2 SYSTEM SEGMENT LEVEL

.1 Segment Requirements Definition
.2 Segment Requirements Evaluation
.3 Segment Design
.4 Segment Design Verification
.5 Segment Change Proposal Evaluation and ECP Preparation
.6 Segment Requirements Documentation
.7 Segment Design Documentation
.8 Segment Definition Monitoring and Direction

.3 CPCI LEVEL DEFINITION

.1 CPCI Requirements Collection and Definition
.2 CPCI Requirements Evaluation
.3 CPCI Design
.4 CPCI Design Verification
.5 CPCI Change Proposal Evaluation and ECP Preparation
.6 CPCI Requirements Documentation
.7 CPCI Design Documentation
.8 CPCI Definition Monitoring and Direction

LEVEL 3 (Con't)

2. CODING

.1 CPCI LEVEL

.1 CPCI Level Coding
.2 CPCI Code Documentation
.3 Monitoring and Direction of CPCI Coding

.2 CPRC LEVEL

.1 CPRC Level Coding, Compilation, and Informal Review

3. DATA CONVERSION

.1 CPCI LEVEL

.1 CPCI Data Conversion
.2 Data Conversion Documentation
.3 Monitoring and Direction of CPCI Data Conversion

4. INFORMAL TEST AND INTEGRATION

.1 CPCI LEVEL

.1 Informal CPCI Test and Integration Planning
.2 Informal CPCI Test and Integration Conduct
.3 Documentation of Informal Test and Integration
.4 Informal CPCI Test and Integration Monitoring and Direction

.2 CPRC LEVEL

.1 CPRC Test and Integration Planning
.2 CPRC Test and Integration Conduct
.3 Documentation of CPRC Test and Integration
.4 CPRC Test and Integration Monitoring and Directives

LEVEL 3 (Con't)

5. FORMAL TEST AND INTEGRATION

.1 SYSTEM LEVEL DT&E

.1 System DT&E Planning
.2 System DT&E Procedure Development
.3 System DT&E Execution
.4 System DT&E Data Reduction
.5 System DT&E Error Identification
.6 System DT&E Documentation
.7 System DT&E Monitoring and Direction

.2 SYSTEM SEGMENT LEVEL DT&E

.1 Segment Test Planning
.2 Segment Test Procedure Development
.3 Segment Test Execution
.4 Segment Test Data Reduction
.5 Segment Test Error Identification
.6 Segment Test Documentation
.7 Segment Test Monitoring and Direction

.3 CPCI LEVEL DT&E

.1 CPCI Qualification Test Planning
.2 CPCI Qualification Test Procedure Development
.3 CPCI Qualification Test Execution
.4 CPCI Qualification Test Data Reduction
.5 CPCI Qualification Test Error Identification
.6 CPCI Qualification Test Documentation
.7 CPCI Qualification Test Monitoring and Direction

6. INSTALLATION

.1 SYSTEM LEVEL

.1 Planning for Installation
.2 Site-Specific Adaptation
.3 Installation Documentation
.4 Installation Monitoring and Direction

LEVEL 3 (Con't)

7. DEVELOPMENT FACILITIES

.1 SYSTEM LEVEL

.1 Development Facility Planning
.2 Development Facility Site Preparation
.3 Development Facility Equipment Acquisition
.4 Development Facility Equipment Maintenance
.5 Development Facility Software Acquisition
.6 Development Facility Software Maintenance and Modification
.7 Development Facility Operation
.8 Development Facility Documentation
.9 Monitoring and Direction of Development Facility Provision and Operation

.2 SYSTEM SEGMENT LEVEL

8. TRAINING

.1 SYSTEM LEVEL

.1 System Training Planning
.2 System Training Material Development
.3 Instruction in System Use, Operation, and Maintenance
.4 System Training Documentation
.5 Monitoring and Direction of System Training

.2 CPCI LEVEL

.1 CPCI Training Material Development
.2 Instruction in CPCI Use, Operation, and Maintenance
.3 CPCI Training Documentation
.4 Monitoring and Direction of CPCI Training

LEVEL 3 (Con't)

9. MANAGEMENT

.1 SYSTEM LEVEL

- .1 System Management Planning
- .2 System Project Direction
- .3 System Configuration Management
- .4 Reporting System Development Status

.2 SYSTEM SEGMENT LEVEL

- .1 Segment Management Planning
- .2 Segment Development Direction
- .3 Segment Configuration Management
- .4 Reporting Segment Development Status

.3 CPCI LEVEL

- .1 CPCI Management Planning
- .2 CPCI Development Direction
- .3 CPCI Configuration Management
- .4 Reporting CPCI Development Status

APPENDIX C

MODEL ESTIMATING PERFORMANCE

TABLE C-1

MODEL ESTIMATING PERFORMANCE - AEROSPACE CORPORATION, COMMERICAL

MODEL: AEROSPACE CORPORATION

DATA SET: COMMERCIAL

CASE	MH ACTUAL	MH* EST	ACT/EST
1. A1	127.2	190.9	0.666
2. A2	38.0	130.5	0.291
3. A3	48.7	163.3	0.298
4. A4	29.3	115.2	0.237
5. A5	45.5	189.6	0.240
6. A6	44.9	216.2	0.208
7. A7	30.5	118.2	0.258
8. A8	53.0	169.5	0.313
9. A9	232.8	307.1	0.758
10. A10	211.1	274.6	0.769
11. A11	13.8	119.2	0.116
Mean	79.5		0.378
Standard Deviation	76.2		0.234

RMS ERROR: 107.

RELATIVE RMS ERROR: 1.35

* Support software relationship (See Appendix A)

TABLE C-2

MODEL ESTIMATING PERFORMANCE - AEROSPACE CORPORATION, DSDC

MODEL: AEROSPACE CORPORATION

DATA SET: DSDC

CASE	MM ACTUAL	MM * EST	ACT / EST
1. DC	2.3	79.4	0.0290
2. DK	79.9	134.8	0.583
3. DS	8.8	101.5	0.0867
4. DU	2.9	95.5	0.0304
5. FB	88.9	169.1	0.526
6. FD	7.1	134.8	0.0527
7. FE	4.9	125.2	0.3091
8. FF	2.6	103.7	0.0251
9. BH	17.8	144.2	0.123
10. BB	9.5	105.4	0.0901
11. GG	48.8	122.3	0.399
12. BI	3.5	76.4	0.0458
13. ZP	9.8	81.0	0.121
14. US	172.9	190.0	0.910
15. JD	45.4	111.4	0.408
16. QD	23.0	153.3	0.150
17. DJ	247.5	104.3	2.37
Mean	45.6		0.353
Standard Deviation	68.8		0.579

RMS ERROR: 96.0

RELATIVE RMS ERROR: 2.11

* Support software relationship (See Appendix A)

TABLE C-3

MODEL ESTIMATING PERFORMANCE - AEROSPACE CORPORATION, SEL

MODEL: AEROSPACE CORPORATION

DATA SET: SEL

CASE	MM ACTUAL	MM* EST	ACT / EST
1. AA	111.0	209.3	0.530
2. AB	221.3	182.7	1.21
3. AC	254.3	209.3	1.22
4. AD	268.8	187.9	1.43
5. AE	324.4	239.9	1.35
6. AF	77.6	160.5	0.484
7. AG	53.1	119.5	0.444
8. AH	29.0	117.8	0.246
9. AI	79.8	157.0	0.508
10. AT	19.5	84.7	0.230
11. AK	25.3	79.0	0.320
12. AL	20.5	103.0	0.199
13. AM	13.9	60.7	0.229
14. AN	23.5	8.31	0.283
15. AO	276.8	218.5	1.27
16. AP	67.0	119.1	0.563
17. AQ	43.8	104.7	0.418
Mean	112.		0.643
Standard Deviation	109.		0.450

RMS ERROR: 67.8

RELATIVE RMS ERROR: 0.605

* Support software relationship (See Appendix A)

TABLE C-4

MODEL ESTIMATING PERFORMANCE - BOEING COMPUTER SERVICES, DSDC

MODEL: BOEING COMPUTER SERVICES

DATA SET: DSDC

CASE	MM ACTUAL	MM EST	$\frac{\text{ACT}}{\text{EST}}$
1. DC	2.4	4.4	0.542
2. DK	82.7	15.0	5.52
3. DS	9.1	7.6	1.19
4. DU	3.0	6.5	0.465
5. FB	92.0	30.8	2.98
6. FD	7.4	13.5	0.550
7. FE	5.1	16.0	0.319
8. FF	2.7	7.8	0.345
9. BH	18.4	24.1	0.764
10. BB	9.8	9.8	0.997
11. GG	50.5	14.8	3.42
12. BI	3.6	3.7	0.983
13. ZP	10.1	4.7	2.14
14. US	178.9	161.2	1.11
15. JD	47.0	10.5	4.48
16. QD	23.8	31.4	0.757
Mean	34.2		1.66
Standard Deviation	48.1		1.60

RMS ERROR: 26.9

RELATIVE RMS ERROR: 0.787

TABLE C-5

MODEL ESTIMATING PERFORMANCE - DOD MICRO PROCEDURE, DSDC

MODEL: DOD MICRO ESTIMATING PROCEDURE

DATA SET: DSDC

CASE	MM ACTUAL	MM EST	ACT/EST
1. DC	2.4	2.7	0.882
2. DK	82.7	14.0	5.90
3. DS	9.1	4.4	2.06
4. DU	3.0	7.7	0.391
5. FB	92.0	26.0	3.54
6. FD	7.4	2.2	3.29
7. FE	5.1	99.6	0.0512
8. FF	2.7	6.5	0.414
9. BH	18.4	74.5	0.247
10. BB	9.8	9.3	1.05
11. GG	50.5	9.6	5.24
12. BI	3.6	1.9	1.89
13. ZP	10.1	6.8	1.49
14. US	178.9	121.7	1.47
15. JD	47.0	6.0	7.89
16. QD	23.8	68.6	0.347
Mean	34.2		2.26
Standard Deviation	48.1		2.32

RMS ERROR: 43.2

RELATIVE RMS ERROR: 1.26

TABLE C-6

MODEL ESTIMATING PERFORMANCE - DOTY ASSOCIATES, INC, DSDC

MODEL: DOTY ASSOCIATES, INC.

DATA SET: DSDC

CASE	MM ACTUAL	MM * EST	$\frac{\text{ACT}}{\text{EST}}$
1. DC	1.0	7.1	0.141
2. DK	32.6	23.2	1.41
3. DS	4.9	12.3	0.398
4. DU	1.1	10.7	0.103
5. FB	38.8	38.5	1.01
6. FD	3.9	23.2	6.00
7. FE	2.4	19.6	0.122
8. FF	1.7	12.9	0.131
9. BH	6.0	26.9	0.223
10. BB	2.8	13.4	0.209
11. GG	8.9	18.6	0.479
12. BI	1.9	6.5	0.292
13. ZP	5.7	7.4	0.770
14. US	48.3	50.0	0.966
15. JD	21.6	15.1	1.43
16. QD	0.8	30.9	0.0259
Mean	11.4		0.725
Standard Deviation	15.3		0.966

RMS ERROR: 12.0

RELATIVE RMS ERROR: 1.05

* Business programs relationship (See Appendix A)

TABLE C-7

MODEL ESTIMATING PERFORMANCE - FARR AND ZAGORSKI, DSDC

MODEL: FARR AND ZAGORSKI

DATA SET: DSDC

CASE	MM ACTUAL	MM * EST	ACT / EST
1. DC	1.0	100.	0.0100
2. DK	32.6	230.	0.142
3. DS	4.9	147.	0.0333
4. DU	1.1	131.	0.00840
5. FB	38.8	343.	0.113
6. FD	3.9	208.	0.0188
7. FE	2.4	163.	0.0147
8. FF	1.7	140.	0.0121
9. BH	6.0	266.	0.0226
10. BB	2.8	155.	0.0181
11. GG	8.9	192.	0.0464
12. BI	1.9	94.	0.0202
13. ZP	5.7	106.	0.0538
14. US	48.3	464.	0.104
15. JD	21.6	178.	0.121
16. QD	0.8	285.	0.00281
17. DJ	21.2	147.	0.144
Mean	12.0		0.0521
Standard Deviation	15.0		0.0509

RMS ERROR: 203.

RELATIVE RMS ERROR: 16.9

* Relationship (3) (See Appendix A)

TABLE C-8

MODEL ESTIMATING PERFORMANCE - PRICE S, COMMERCIAL

MODEL: PRICE S

DATA SET: COMMERCIAL

CASE	MM ACTUAL	MM EST	ACT / EST
1. A1	87.2	48	1.82
2. A2	22.9	17	1.35
3. A3	38.8	44	0.882
4. A6	37.2	71	0.524
5. A8	40.7	12	3.39
6. A9	192.0	176	1.09
7. A11	10.5	17	0.618
Mean	61.3		1.38
Standard Deviation	62.3		0.990

RMS ERROR: 23.5

RELATIVE RMS ERROR: 0.383

TABLE C-9

MODEL ESTIMATING PERFORMANCE - PRICE S, DSDC

MODEL: PRICE S

DATA SET: DSDC

CASE	MM ACTUAL	MM EST	ACT / EST
1. DC	1.7	36	.0472
2. DU	2.4	8	.300
3. FD	5.5	11	.500
4. FE	3.8	14	.271
5. FF	2.0	9	.222
6. BH	14.5	19	.763
7. BB	7.4	10	.740
8. GG	40.2	12	3.35
9. ZP	8.0	6	1.33
10. US	129.5	35	3.70
11. JD	35.2	11	3.20
12. QD	11.7	22	.532
Mean	21.8		1.25
Standard Deviation	36.2		1.35

RMS ERROR: 31.4

RELATIVE RMS ERROR: 1.44

TABLE C-10

MODEL ESTIMATING PERFORMANCE - PRICE S, SEL

MODEL: PRICE S

DATA SET: SEL

CASE	MM ACTUAL	MM EST	ACT/EST
1. AA	39.6	48	0.825
2. AB	79.0	85	0.929
3. AC	90.7	96	0.945
4. AD	95.9	78	1.23
5. AE	115.7	150	0.771
6. AF	27.7	27	1.03
7. AG	18.9	22	0.860
8. AH	10.3	22	0.470
9. AI	28.5	39	0.730
10. AJ	7.0	5	1.39
11. AK	9.0	10	0.902
12. AL	7.3	16	0.458
13. AN	8.4	11	0.764
Mean	41.1		0.870
Standard Deviation	39.4		0.260

RMS ERROR: 12.3

RELATIVE RMS ERROR: 0.297

TABLE C-11

MODEL ESTIMATING PERFORMANCE - SLIM, COMMERCIAL

MODEL: SLIM

DATA SET: COMMERCIAL

CASE	MM ACTUAL	MM EST	ACT / EST
1. A1	71	38.5	1.84
2. A3	38	20.7	1.84
3. A5	32	37.1	0.863
4. A6	36	34.0	1.06
5. A9	184	212.1	0.868
6. A10	163	138.5	1.18
Mean	87.3		1.27
Standard Deviation	68.5		0.454

RMS ERROR: 21.5

RELATIVE RMS ERROR: 0.246

TABLE C-12

MODEL ESTIMATING PERFORMANCE - SLIM, DSDC

MODEL: SLIM

DATA SET: DSDC

CASE	MM ACTUAL	MM EST	ACT/EST
1. FB	81.3	79.8	1.02
2. BH	17.4	27.1	0.642
3. US	155.6	125.4	1.24
Mean	84.8		0.967
Standard Deviation	69.2		0.303

RMS ERROR: 18.3

RELATIVE RMS ERROR: 0.216

TABLE C-13

MODEL ESTIMATING PERFORMANCE - SLIM, SEL

MODEL: SLIM

DATA SET: SEL

CASE	MM ACTUAL	MM EST	ACT/EST
1. AA	39.6	45.1	0.878
2. AB	79.0	76.9	1.03
3. AC	90.7	90.0	1.01
4. AD	95.9	102.7	0.934
5. AE	115.7	307.9	0.376
6. AI	28.5	44.6	0.639
7. A6	138.3	148.2	0.933
8. A7	98.4	179.7	0.548
Mean	85.8		
Standard Deviation	36.6		

RMS ERROR: 74.2

RELATIVE RMS ERROR: 0.865

TABLE C-14

MODEL ESTIMATING PERFORMANCE - TECOLOTE, DSDC

MODEL: TECOLOTE

DATA SET: DSDC

CASE	MM ACTUAL	MM* EST	$\frac{ACT}{EST}$
1. DC	2.4	38.1	0.0629
2. DK	82.7	193.1	0.428
3. DS	9.1	80.8	0.113
4. DU	3.0	67.1	0.0447
5. FB	92.0	387.4	0.238
6. FD	7.4	193.3	0.0383
7. FE	5.1	153.9	0.0331
8. FF	2.7	86.4	0.0313
9. BH	18.4	237.7	0.0774
10. BB	9.8	90.7	0.108
11. GG	50.5	143.2	0.353
12. BI	3.6	33.8	0.106
13. ZP	10.1	40.5	0.250
14. US	178.9	554.5	0.323
15. JD	47.0	107.6	0.437
16. QD	23.8	286.7	0.0830
Mean	34.2		0.170
Standard Deviation	48.1		0.580

RMS ERROR: 168.

RELATIVE RMS ERROR: 4.92

* Estimating relationship using number of operating instructions.

TABLE C-15

MODEL ESTIMATING PERFORMANCE - WOLVERTON, DSDC

MODEL: WOLVERTON

DATA SET: DSDC

CASE	MM ACTUAL	MM EST	ACT/EST
1. DC	2.4	3.4	0.712
2. DK	82.7	45.4	1.82
3. DS	9.1	6.4	1.43
4. DU	3.0	24.8	0.121
5. FB	92.0	124.7	0.738
6. FD	7.4	16.8	0.441
7. FE	5.1	45.5	0.112
8. FF	2.7	29.7	0.0910
9. BH	18.4	11.6	1.59
10. BB	9.8	23.2	0.422
11. GG	50.5	23.6	2.14
12. BI	3.6	8.1	0.445
13. ZP	10.1	6.9	1.46
14. US	178.9	106.5	1.68
15. JD	47.0	17.7	2.65
16. QD	23.8	48.6	0.49
Mean	34.2		1.02
Standard Deviation	48.1		0.802

RMS ERROR: 31.7

RELATIVE RMS ERROR: 0.927

TABLE C-16

MODEL ESTIMATING PERFORMANCE - RECALIBRATED SIZE EQUATION

MODEL: RECALIBRATED SIZE EQUATION ($MM = aI^b$)

COMMERCIAL a = 0.293 b = 0.502					D S D C a=1.69x10-4 b=1.28				S - L a=7.21x10-3 b=0.897			
	SYS	MM ACT	MM EST	ACT/EST	SYS	MM ACT	MM EST	ACT/EST	SYS	MM ACT	MM EST	ACT/EST
1.	A1	71	51.0	1.39	DC	2.0	3.1	0.645	AA	39.60	36.2	1.09
2.	A2	16	29.3	0.546	DK	66.8	21.7	3.08	AB	78.95	63.3	1.25
3.	A3	38	40.6	0.936	DS	8.5	7.6	1.12	AC	90.71	67.3	1.35
4.	A4	17	24.5	0.694	DU	2.9	6.1	0.475	AD	95.90	60.4	1.59
5.	A5	32	50.5	0.634	FB	81.3	50.1	1.62	AE	115.71	122.7	0.943
6.	A6	36	61.1	0.589	FD	6.5	21.7	0.300	AF	27.69	45.9	0.603
7.	A7	24	9.6	2.50	FE	4.5	16.5	0.273	AG	18.93	19.9	0.951
8.	A8	38	23.1	1.65	FF	2.4	8.2	0.293	AH	10.33	15.7	0.658
9.	A9	184	101.8	1.81	BH	17.4	27.8	0.626	AI	28.48	35.8	0.796
10.	A10	163	86.5	1.88	BB	8.8	8.7	1.01	AJ	6.96	7.2	0.967
11.	A11	10	25.7	0.389	GG	48.3	15.1	3.20	AK	9.02	6.9	1.31
12.					BI	3.5	2.7	1.30	AL	7.32	14.1	0.519
13.					ZP	9.7	3.3	2.94	AM	4.96	3.7	1.34
14.					US	155.6	77.1	2.02	AN	8.40	7.9	1.06
15.					JD	42.3	10.7	3.95	AO	98.74	99.2	0.995
16.					QD	14.0	34.9	0.401	AP	23.92	19.2	1.25
17.									AQ	15.63	15.1	1.04
Mean		57.2		1.18		29.7		1.45		40.1		1.04
Standard Deviation		59.9		0.697		41.8		1.23		38.9		0.289
RMS ERROR:				36.8				27.7				12.4
RELATIVE RMS ERROR:				0.643				0.933				0.309

TABLE C-17
SUMMARY OF MODEL ESTIMATING PERFORMANCE

RMS ERROR*
MEAN PROJECT SIZE

MODEL TYPE	DATA SET		
	COMMERCIAL	D S D C	S E L
REGRESSION			
A AEROSPACE	107.	96.0	67.8
B DOTY		12.0	
C FARR & ZAGORSKI		203.	
D TECOLOTE		168.	
E (aI^b)	36.8	27.7	12.4
HEURISTIC			
F BOEING		26.9	
G DOD MICRO		43.2	
H PRICE S	23.5	31.4	12.3
I WOLVERTON		31.7	
PHENOMENOLOGICAL			
J SLIM	21.5	18.3	74.2

* RMS ERROR = $\left[\frac{1}{N}\sum_{i=1}^{N}(ACT_i - EST_i)^2\right]^{1/2}$

APPENDIX D

DESCRIPTION OF MODEL INPUTS

TABLE D-1

AEROSPACE AND TECOLOTE MODELS

SUMMARY OF INPUTS

	COMMERCIAL		DSDC		SEL	
	SYS.	OBJ. INS.	SYS.	OBJ. INS.	SYS.	OBJ. INS.
1.	A1	78335	DC	8943	AA	98402
2.	A2	30543	DK	33090	AB	70306
3.	A3	53200	DS	16390	AC	98402
4.	A4	22459	DU	14110	AD	75357
5.	A5	77039	FB	58010	AE	137944
6.	A6	106573	FD	33120	AF	51030
7.	A7	23939	FE	27560	AG	24575
8.	A8	58391	FF	17290	AH	23726
9.	A9	254232	BH	39120	AI	48273
10.	A10	192706	BB	17990	AJ	10476
11.	A11	24420	GG	26000	AK	8823
12.			BI	8116	AL	17022
13.			ZP	9383	AM	4596
14.			US	77470	AN	9985
15.			JD	20640	AO	109466
16.			QD	45510	AP	24392
17.			DJ	17560	AQ	17725

TABLE D-2

BOEING COMPUTER SERVICES

SUMMARY OF INPUTS

DSDC DATA

SYSTEM	REPT $\hat{E}$	STMTS $\hat{S}$	LOGIC $\hat{E}$	STMTS $\hat{S}$	PROG. $\hat{E}$	EXP. $\hat{S}$	NO. PROGS.	EXIST. SW	F.O.	HOL	MACRO-CODE	MACRO-FORMS	ON-LINE DEBUG	NO DEBUG TOOLS	ON-LINE CODE/DATA ENTRY
1. DC	1751		382		5.0	3.7	1	N	N	Y	N	N	N	N	N
2. DK	6071		1126		5.0	3.7	2								
3. DS	3500		305		5.0	3.7	1								
4. DU	2773		348		5.0	3.7	1								
5. FB	14421		2394		6.9		2								
6. FD	2814		547		8.6		3								
7. FE	6512		1827		9.9		1								
8. FF	3887		444		9.6		1								
9. BH	8893	497.5	1477	281.5	1.6		1								
10. BB	3665	485.3	594	274.7	2.8		1								
11. GG	6057		1056		5.0	3.7	2								
12. BI	1526	491.2	232	278.0	5.0	3.7	1								
13. ZP	1783	490.1	276	277.4	2.6		1								
14. US	19604	621.9	3288	352.0	0.7		4								
15. JD	4703		485		5.0	3.7	2								
16. QD	11279		2151		1.9		1								
17. DJ	3568	485.4	577	274.7	5.0	3.7	5								

TABLE D-3

MICRO ESTIMATING PROCEDURE

SUMMARY OF INPUTS

DSDC DATA

SYSTEM	NO. I/O FILES		COMPLEXITY		EXP. AVAIL		JOB KNOW.R		JOB KNW AV.		TURN AROUND		SYS. FCTR	
	$\hat{E}$	$\hat{S}$	$\hat{E}$	$\hat{S}$	$\hat{E}$	$\hat{S}$	$\hat{E}$	$\hat{S}$	$\hat{E}$	$\hat{S}$	$\hat{E}$	$\hat{S}$	$\hat{E}$	$\hat{S}$
1. DC	59		144.5		1.08	0.500	0.5		0.72	0.363	0.8		3	
2. DK	308		209.5		1.08	0.500	1.0		0.72	0.363	0.8		3	
3. DS	158		168		1.08	0.500	0.5		0.72	0.363	0.8		3	
4. DU	151		129		1.08	0.500	1.0		0.72	0.363	0.8		3	
5. FB	709		536.5		0.75		1.0		0.5		0.8		3	
6. FD	220		138.5		0.75		0.5		0.5		0.8		5	
7. FE	185		289		0.75		1.0		0.5		0.8		3	
8. FF	143		168.5		0.75		1.0		0.5		0.8		3	
9. BH	386	48.4	390	70.3	1.75		1.0		1.0		0.8		3	
10. BB	161	47.2	177	68.6	0.75		1.5		0.5		0.8		3	
11. GG	280		325		1.08	0.500	0.89	0.350	0.72	0.363	0.8		3	
12. BI	69	48.0	90	69.7	1.08	0.500	0.89	0.350	0.72	0.363	0.8		3	
13. ZP	80	47.7	100	69.2	1.75		1.0		0.5		0.8		3	
14. US	847	60.5	826	87.9	1.75		0.5		1.5		0.8		3	
15. JD	262		178		1.08	0.500	0.5		0.72	0.363	0.8		3	
16. QD	480		617.5		0.75		1.5		1.0		0.8		3	
17. DJ	157	47.2	173	68.6	1.08	0.500	0.89	0.350	0.72	0.363	0.8		3	

TABLE D-4

DOTY

SUMMARY OF INPUTS

DSDC

	SYS	SOURCE LINES	f_1^*	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
1.	DC	2286	N	Y	N	N	N	N	N	N	Y	N	Y	N	N	N
2.	DK	10550	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
3.	DS	4641	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
4.	DU	3897	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
5.	FB	20334	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
6.	FD	10559	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
7.	FE	8519	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
8.	FF	4940	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
9.	FH	12829	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
10.	BB	5174	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
11.	GG	7958	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
12.	BI	2041	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
13.	ZP	2418	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
14.	US	28514	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
15.	JD	6077	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
16.	QD	15310	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓

* Definitions of the parameters are given in Appendix A.

TABLE D-5

FARR & ZAGORSKI MODEL

SUMMARY OF INPUTS

DSDC DATA

	SYSTEM	x_{11}	x_{12}	x_5	x_3	x_{13}
1.	DC	4.655	3.674	2.788	6	3.6
2.	DK	16.427	14.010	3.424	6	3.6
3.	DS	8.910	6.080	3.146	6	3.6
4.	DU	7.326	5.721	3.027	6	3.6
5.	FB	27.538	25.328	3.711	6	5.2
6.	FD	14.865	16.032	3.347	6	6.6
7.	FE	9.569	15.996	3.300	6	6.3
8.	FF	9.273	7.061	3.200	6	6.1
9.	BH	19.064	16.764	3.517	6	1.5
10.	BB	8.909	7.626	3.163	6	2.3
11.	GG	12.733	11.196	3.316	6	3.6
12.	BI	4.001	3.518	2.776	6	3.6
13.	ZP	4.616	4.060	2.849	6	2.6
14.	US	37.696	33.148	3.821	6	0.0
15.	JD	11.135	7.276	3.348	6	1.9
16.	QD	21.242	20.304	3.598	6	3.6

TABLE D-6

PRICE S
SUMMARY OF INPUTS

COMMERCIAL

	SYSTEM	INST	APPL	RESO	UTIL	PLTFM	CPLX	NEWD	NEWC	COST
1.	A1	142732	3.00	1.88	0.50	1.00	1.00	0.78	0.83	4000
2.	A2	45200	3.00	1.88	0.50	1.00	1.00	0.70	0.80	4000
3.	A3	148700	3.00	1.88	0.50	1.00	1.00	0.90	0.50	4000
4.	A6	194748	3.00	1.88	0.50	1.00	1.00	0.90	1.00	4000
5.	A8	2800	3.00	1.88	0.50	1.00	1.00	0.90	0.70	4000
6.	A9	585200	3.00	1.88	0.50	1.00	1.00	0.85	0.95	4000
7.	A11	38800	3.00	1.88	0.50	1.00	1.00	0.80	1.00	4000

TABLE D-7

PRICE S

SUMMARY OF INPUTS

DSD-

	SYSTEM	INST	APPL	RESO	UTIL	PLTFM	CPLX	NEWD	NEWC	COST
1.	DC	8943	2.45	2.13	0.50	1.00	1.00	0.90	1.00	4000
2.	DU	14110	2.58	2.13	0.50	1.00	1.00	0.90	1.-0	4000
3.	FD	33120	2.56	2.13	0.50	1.00	1.00	0.20	1.00	4000
4.	FE	27560	2.50	2.13	0.50	1.00	1.00	1.00	1.00	4000
5.	FF	17290	2.43	2.13	0.50	1.00	1.00	1.00	1.00	4000
6.	BH	39.20	2.51	2.13	0.50	1.00	1.00	1.00	1.00	4000
7.	BB	17990	2.51	2.13	0.50	1.00	1.00	1.00	1.00	4000
8.	GG	26000	2.36	2.13	0.50	1.00	1.00	1.00	1.00	4000
9.	ZP	9383	2.51	2.13	0.50	1.00	1.00	1.00	1.00	4000
10.	US	77470	2.51	2.13	0.50	1.00	1.00	1.00	1.00	4000
11.	JD	20640	2.51	2.13	0.50	1.00	1.00	1.00	1.00	4000
12.	QD	45510	2.51	2.13	0.50	1.00	1.00	1.00	1.00	4000

TABLE D-8

PRICE S

SUMMARY OF INPUTS

SEL

	SYSTEM	INST	APPL	RESO	UTIL	PLTFM	CPLX	NEWD	NEWC	COST
1.	AA	150800	5.71	2.25	0.80	1.20	0.60	0.20	0.30	4000
2.	AB	101800	5.73	2.25	0.80	1.20	0.60	0.90	0.90	4000
3.	AC	150800	5.65	2.25	0.80	1.20	0.60	0.60	0.70	4000
4.	AD	110400	5.52	2.25	0.80	1.20	0.60	0.70	0.80	4000
5.	AE	223800	5.55	2.25	0.80	1.20	0.60	0.70	0.80	4000
6.	AG	29800	5.99	2.25	0.80	1.20	0.60	0.80	0.80	4000
7.	AH	28600	5.44	2.25	0.80	1.20	0.60	0.80	0.70	4000
8.	AI	65600	5.58	2.25	0.80	1.20	0.60	0.50	0.60	4000
9.	AJ	11000	3.53	2.25	0.80	1.20	0.60	0.30	0.50	4000
10.	AK	9000	5.52	2.25	0.80	1.20	0.60	1.00	0.90	4000
11.	AL	19400	5.42	2.25	0.80	1.20	0.60	0.60	0.80	4000
12.	AN	10400	5.99	2.25	0.80	1.20	0.60	0.90	0.90	4000

TABLE D-9

SLIM

SUMMARY OF INPUTS

	COMMERCIAL				DSDC				SEL			
	SYSTEM	STMTS	LEVEL	TECH. FACTOR	SYSTEM	STMTS	LEVEL	TECH. FACTOR	SYSTEM	STMTS	LEVEL	TECH. FACTOR
1.	A1	32276	3	15	FB	19845	3	9	AA	13400	5	11
2.	A3	20534	4	15	BH	12381	2	9	AB	25000	3	11
3.	A5	31653	3	15	US	27519	2	9	AC	26800	3	11
4.	A6	46253	1	15					AD	23750	4	11
5.	A9	127775	2	15					AE	52350	3	11
6.	A10	92415	2	15					AI	13250	5	11
7.									A6	52983	1	11
8.									A7	37900	3	11

TABLE D-10

WOLVERTON MODEL

SUMMARY OF INPUTS

DSDC DATA

		DIFF.		OBJECT INSTS		C		I		A		D	
		$\bar{X}$	S_X	$\bar{X}$	S_X	$\bar{X}$	S_X	$\bar{X}$	S_X	$\bar{X}$	S_X	$\bar{X}$	S_X
1.	DC	1.8		8,943	10,570	55.89		.88		20.07		3.16	
2.	DK	3.3		33,090	41,578	53.97		3.59		19.55		22.89	
3.	DS	1.5		16,390	19,816	59.44		3.60		13.05		23.91	
4.	DU	3.3		14,110	16,950	56.15		2.69		15.95		25.21	
5.	FB	2.3		58,010	75,910	52.09		4.44		19.78		23.69	
6.	FD	1.7		33,120	41,620	48.11		1.76		20.73		29.40	
7.	FE	3.1		27,560	34,260	37.43		1.45		28.15		32.97	
8.	FF	2.5		17,290	20,950	56.77		2.21		20.43		20.59	
9.	BH	3.1		39,120	49,740	53.21	6.34	2.77	1.26	19.81	15.10		
10.	BB	4.2		17,990	21,840	53.21	6.34	2.77	1.26	19.81	15.10		
11.	GG	2.7	.865	26,000	32,200	53.88		1.95		23.73		20.44	
12.	BI	2.7	.865	8,116	9,568	53.21	6.34	2.77	1.26	19.81	15.10		
13.	ZP	2.6		9,383	11,110	53.21	6.34	2.77	1.26	19.81	15.10		
14.	US	2.2		77,470	103,800	53.21	6.34	2.77	1.26	19.81	15.10		
15.	JD	2.0		20,640	25,240	60.48		3.00		15.98		20.54	
16.	QD	4.1		45,510	58,460	51.13		4.85		20.51		23.51	
17.	DJ	2.7	.865	17,560	21,300	53.21	6.34	2.77	1.26	19.81	15.10		

TABLE D-11

RECALIBRATED SIZE EQUATION

SUMMARY OF INPUTS

	COMMERCIAL		DSDC		SEL	
	SYSTEM	STMTS	SYSTEM	STMTS	SYSTEM	STMTS
1.	A1	28879	DC	2205	AA	13400
2.	A2	9605	DK	10182	AB	25000
3.	A3	18373	DS	4479	AC	26800
4.	A4	6706	DW	3761	AD	23750
5.	A5	28321	FB	19624	AE	52350
6.	A6	41384	FD	10190	AF	17500
7.	A7	1046	FE	8222	AG	6900
8.	A8	5950	FF	4768	AH	5300
9.	A9	114325	BH	12381	AI	13250
10.	A10	82687	BB	4993	AJ	2200
11.	A11	7395	GG	7680	AK	2100
12.			BI	1970	AL	4700
13.			ZP	2334	AM	1050
14.			US	27519	AN	2450
15.			JD	5865	AO	41300
16.			QD	14776	AP	6606
17.					AQ	5077

REFERENCES

1. Major System Acquisitions, Dept. of Defense Directive 5000.1, Jan. 18, 1977.

2. Major System Acquisition Process, Dept. of Defense Directive 5000.2, Jan. 18, 1977.

3. Economic Analysis and Program Evaluation for Resource Management, Air Force Regulation 1978-1, Dec. 28, 1973.

4. Computer Technology Forecast and Weapon Systems Impact Study (COMTEC-2000), Vol. II, Technical Data, NTIS, AD B034955L, Dec 1978.

5. A. Ash, D. W. Kelliher, J. P. Locker, III, T. Connors, DoD Weapon Systems Software Acquisition and Management Study, Vol. I, MITRE Findings and Recommendations, MITRE Corporation, MTR-6908, May 1975.

6. T. G. James, Jr., Software Cost Estimating Methodology, NAECON 77 Record pp 22-28.

7. David A. Fisher, Automatic Data Processing Cost in the Defense Department, Institute for Defense Analyses, Paper P-1046, Oct 1974.

8. Barry W. Boehm, Software and Its Impact: A Quantitative Assessment, Datamation, May 1974, pp 48-59.

9. E. N. Dodson, E. E. Balkovich, and W. E. Waller, Advanced Cost Estimating and Synthesis Techniques for Avionics, General Research Corporation CR-2-461, Sept 1975.

10. E. N. Dodson, et.al., Cost Estimating Relationships for Airborne Radars FLIRS, and Avionics Logistics Support, General Research Corporation, CR-1-559.

11. M. Eddins Earles, Factors, Formulas and Structures for Life Cycle Costing, Eddins-Earles, Concord, Mass., 1978.

12. P. F. Ostwald, Cost Estimating for Engineering and Management, Prentice-Hall, Englewood Cliffs, New Jersey, 1974.

13. E. N. Dodson, Studies of Improved Techniques for Parametric Cost Analysis, General Research Corporation, IM-1957, Dec 1974.

14. E. H. Yates, Interrelationships of Technology, System Performance, and Prices for Mini/Midi Computers, General Research Corporation, TIO 2286, Aug 1980.

15. J. R. Brown, Impact of MPP on System Development, Rome Air Development Center, RADC-TR-77-121.

16. Summary Notes of a Government/Industry Software Sizing and Costing Workshop, National Technical Information Service, AD-A026964, Oct 1974.

17. W. S. Junk, A Software Cost Estimation Methodology Creating the Structure for Reliable Application of State of the Art Cost Models, Workshop in Quantitative Software Models, IEEE Cat. No. TH0067-9, Oct 1979, pp 56-62.

18. R. L. Smith, Estimating Software Project Resource Requirements, Vol. XI, Structured Programming Series, Rome Air Development Center, RADC-TR-74-300, Vol. XI, Jan 1975.

19. G. L. Meyers, Estimating the Costs of a Programming System Development Project, IBM, Systems Development Division, IBM TR00.2316, May 1972.

20. Quantitative Software Models, Data and Analysis Center for Software, RADC, SRR-1, Mar 1979.

21. T. J. Devenny, An Exploratory Study of Cost Estimating at the Electronics Systems Division, NTIS, AD A030162.

22. L. H. Putnam, R. W. Wolverton, Quantitative Management Software Cost Estimating, IEEE Cat. No. EHO 129-7.

23. W. S. Junk, J. A. McCall, L. H. Putnam, G. F. Walters, Survey of Software Cost Estimating Techniques, General Electric, Information Systems Programs, 78CIS010, May 1978.

24. J. A. Clapp, A Review of Software Cost Estimation Methods, NTIS AD A029748, Aug 1976.

25. M. Finfer, R. Mish, Software Acquisition Management Guidebook: Cost Estimation and Measurement, NTIS, AD A055574, Mar 1978.

26. M. L. Shooman, Tutorial on Software Cost Models, Workshop on Quantitative Software Models, IEEE Cat. No. TH0067-9, Oct 1979, pp 1-19.

27. G. H. Sandler, B. I. Rachowitz, Software Cost Models - Grumman Experience, Workshop on Quantitative Software Models, IEEE Cat. No. TH0067-9, pp 69-77.

28. T. G. James, Jr., D. V. Ferens, Application of the RCA PRICE-S Software Cost Estimation Model to Air Force Avionics Laboratory Programs, AFSC, AF Avionics Laboratory, AFAL-TR-79-1164, Oct 1979.

29. R. E. Steffey, Jr., An Analysis of the RCA PRICE-S Cost Estimation Model as it Relates to Current Air Force Computer Software Acquisition and Management, Thesis, AFIT, GSM/SM/79D-20, Dec 1979.

30. J. Schneider, IV, A Preliminary Calibration of the RCA PRICE S Software Cost Estimation Model, Thesis, NTIS, AD A046808, Sept 1977.

31. J. W. Bailey, V. R. Basili, A Meta-Model for Software Development Resource Expenditures, Dept. of Computer Science, Univ. of Maryland.

32. B. Curtis, Measurement and Experimentation in Software Engineering, Proceedings of the IEEE, Vol. 68, No. 9, Sept 1980, p 1151.

33. V. R. Basili, R. W. Reiter, Jr., An Investigation of Human Factors in Software Development, Computer, Dec 1979, pp 21-38.

34. J. B. Glore, Software Acquisition Management Guidebook: Life Cycle Events, AFSC, Electronic Systems Division, ESD-TR-77-22, Feb 1977.

35. D. R. Peterson, Software Acquisition Management Guidebook: Software Development and Maintenance Facilities, NTIS, AD 038234, Apr 1977.

36. P. V. Norden, Useful Tools for Project Management, Management of Production, M. K. Starr, Ed., Penguin Books, 1970, pp 71-101.

37. P. V. Norden, Project Life Cycle Modelling: Background and Application of the Life Cycle Curves, Software Life Cycle Management Workshop, US Army Computer Systems Command, Aug 1977, pp 217-306.

38. L. H. Putman, A General Empirical Solution to the Macro Software Sizing and Estimating Problem, IEEE Transactions on Software Engineering, Vol. SE-4, No. 4, July 1978, pp 345-361.

39. M. H. Halstead, Elements of Software Science, Elsevier North-Holland, 1977.

40. L. A. Belady, M. M. Lehman, The Evolution Dynamics of Large Programs, IBM Watson Research Center, Sept 1975.

41. R. Thibodeau, E. N. Dodson, Life Cycle Phase Interrelationships, Journal of Systems and Software, Vol 1, No. 3, 1980, pp 203-211.

42. S. S. Yau, J. S. Collofello, Performance Ripple Effect Analysis for Large Scale Software Maintenance, AFSC, Rome Air Development Center, RADC-TR-80-55, Mar 1980.

43. C. A. Graver, E. E. Balkovich, W. M. Carriere, R. Thibodeau, Cost Reporting Elements and Activity Cost Tradeoffs for Defense Systems Software, General Research Corporation, CR-721, Mar 1977.

44. W. M. Carriere, R. Thibodeau, Development of a Logistics Software Cost Estimating Technique for Foreign Military Sales, General Reserach Corporation, CR-3-839, Jun 1979.

45. T. Yamane, Statistics, An Introductory Analysis, Harper & Row, 2nd Ed, 1967, pp 709-711.

46. F. P. Brooks, The Mythical Man-Month, Datamation, Dec 1974, pp 45-52.

47. Reference Manual PRICE Software Model, RCA/PRICE Systems.

48. SLIM User's Guide, Quantitative Management Systems, Inc.

49. J. H. Herd, J. N. Postak, W. E. Russell, K. R. Stewart, Software Cost Estimating Study, Study Results, Vol. I, AFSC Rome Air Development Center, RADC-TR-77-220, Jun 1977.

50. E. A. Nelson, Management Handbook for the Estimation of Computer Programming Costs, System Development Corporation, RM-3225/000/01, Mar 1967.

51. R. Thibodeau, The State-of-the-Art in Software Error Data Collection and Analysis, NTIS AD A075228, Jan 1979.

52. R. Thibodeau, The Feasibility of Obtaining Software Research Data at the US Army Computer Systems Command, Army Institute for Research in Management Information and Computer Science (AIRMICS), Jul 1980.

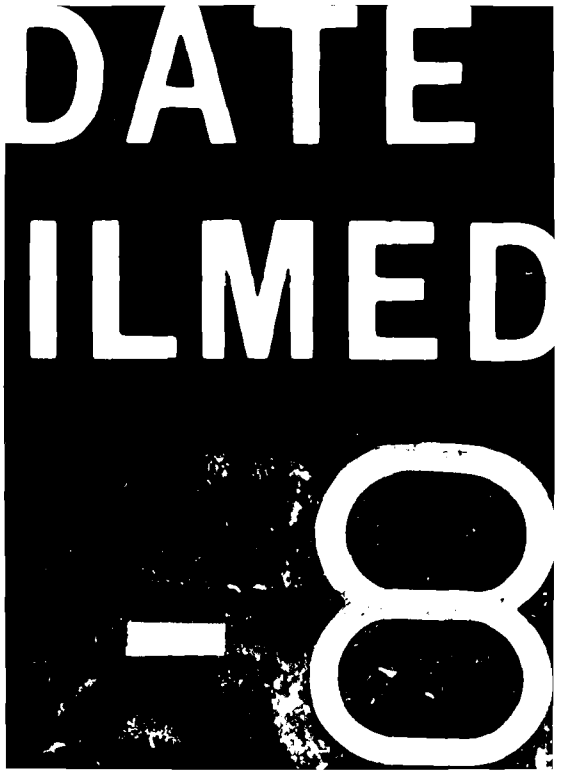
53. S. Pelosi, B. Parham, J. Berterman, Arlen Feldman, Automatic Data Processing Resource Estimating Procedures (ADPREP), NTIS, AD 711117, Aug 1970, p 23.

54. R. W. Wolverton, The Cost of Developing Large-Scale Software, IEEE Transactions on Computers, Jun 1974, p 626.

55. C. E. Walston, Working Group on Software Cost, Workshop Quantitative Software Models, IEEE Cat. No. TH0067-9, Oct 1979, p 241.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. 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.


DATE
ILMED
-8