IDA PAPER P-2388

IMPROVING METHODS FOR ESTIMATING SOFTWARE DEVELOPMENT COSTS

Steven R. Shyman
T. Keith Blankenship

June 1990

Prepared for
Strategic Defense Initiative Organization

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311-1772

IDA Log No. HQ 90-35427

DEFINITIONS

IDA publishes the following documents to report the results of its work.

Reports

Reports are the most authoritative and most carefully considered products IDA publishes. They normally embody results of major projects which (a) have a direct bearing on decisions affecting major programs, (b) address issues of significant concern to the Executive Branch, the Congress and/or the public, or (c) address issues that have significant economic implications. IDA Reports are reviewed by outside panels of experts to ensure their high quality and relevance to the problems studied, and they are released by the President of IDA.

Group Reports

Group Reports record the findings and results of IDA established working groups and panels composed of senior individuals addressing major issues which otherwise would be the subject of an IDA Report. IDA Group Reports are reviewed by the senior individuals responsible for the project and others as selected by IDA to ensure their high quality and relevance to the problems studied, and are released by the President of IDA.

Papers

Papers, also authoritative and carefully considered products of IDA, address studies that are narrower in scope than those covered in Reports. IDA Papers are reviewed to ensure that they meet the high standards expected of refereed papers in professional journals or formal Agency reports.

Documents

IDA Documents are used for the convenience of the sponsors or the analysts (a) to record substantive work done in quick reaction studies, (b) to record the proceedings of conferences and meetings, (c) to make available preliminary and tentative results of analyses, (d) to record data developed in the course of an investigation, or (e) to forward information that is essentially unanalyzed and unevaluated. The review of IDA Documents is suited to their content and intended use.

The work reported in this document was conducted under contract MDA 903 89 C 0003 for the Department of Defense. The publication of this IDA document does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that Agency.

This Paper has been reviewed by IDA to assure that it meets high standards of thoroughness, objectivity, and appropriate analytical methodology and that the results, conclusions and recommendations are properly supported by the material presented.

UNCLASSIFIED

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1990	Final Report, Jun 1989 – Jun 1990

4. TITLE AND SUBTITLE
Improving Methods for Estimating Software Development Costs

5. FUNDING NUMBERS
C-MDA-903-89C-0003
T-R2-597.12

6. AUTHOR(S)
Steven R. Shyman and T. Keith Blankenship

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311-1772

8. PERFORMING ORGANIZATION REPORT NUMBER
IDA-P-2388

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
SDIO/S/PI (SE Design)
Room 1E149, The Pentagon
Washington, D.C. 20301

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Two methods for deriving more accurate estimates of software development costs are explored. The first method involves the use of a simple cost estimating relationship in which the single input of size (in lines of code) is used. The second method involves multiple inputs used in the Constructive Cost Model (COCOMO).

14. SUBJECT TERMS
Cost Models, Cost Estimates, Constructive Cost Model (COCOMO), Computer Programs

15. NUMBER OF PAGES
37

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

UNCLASSIFIED

IDA PAPER P-2388

IMPROVING METHODS FOR ESTIMATING SOFTWARE DEVELOPMENT COSTS

Steven R. Shyman
T. Keith Blankenship

June 1990

INSTITUTE FOR DEFENSE ANALYSES
Contract MDA 903 89 C 0003
Task T-R2-597.12

PREFACE

This paper was prepared by the Institute for Defense Analyses (IDA) for the Strategic Defense Initiative Organization (SDIO), under contract MDA 903 89 C 0003, Subtask Order T-R2-597.12, "Strategic Defense System Phase One Engineering and Technical Support (POET)," issued 1 October 1988. The objective of the subtask was to provide analyses and recommendations to SDIO for defining a baseline concept for the Phase One Strategic Defense System (SDS).

In support of that objective, IDA examined the size and cost estimates for the software involved in the Phase One SDS. Part of that work involved the development of improved ways of implementing the cost estimating methods used to estimate the cost of software. These improved methods are the subject of this paper.

This work was reviewed by Bruce R. Harmon and James Baldo, Jr., of IDA and William Kuhn of MITRE Corporation.

EXECUTIVE SUMMARY

The inaccurate estimates that sometimes result from the use of cost estimating relationships (CERs) come from two sources: (1) inaccuracy of the inputs and (2) inaccuracy of the relationships themselves. The work presented here concentrates on the second of these problems.

Two basic approaches to estimating software development costs were taken, one in which only a single input was used (the single-input case) and one in which multiple inputs were used (the multiple-input case). For the single-input case, we used a simple CER with only size (lines of code) as the input. For the multiple-input case, we used a commercially available cost estimating model that required qualitative inputs as well as size input.

The intent of the single-input case was to show that when a variety of software types (space, avionics, ground) are involved, as in the case of the Phase One Strategic Defense System (SDS), using only one CER yields less accurate estimates than does using several CERs. Our approach was to segregate historical data on software size (measured in lines of code) according to platform and function and derive a CER for each segregated database. Our conclusions for the single-input case were as follows:

- More accurate results can be obtained if the data are segregated by software application. Segregation captures the effect on cost of factors that are difficult to quantify.
- The development of space-based software costs six times more than that of ground-based software, given size.
- Software developed for avionic functions (command, control, and communication (C^3), radar, and integration software residing in air, on sea, or on ground) costs two to four times more than ground-based software that functions in support of space activities. The cost range depends on the size of the software.

The intent of the multiple-input case was to show that predictive ability could be enhanced by adding more information. Using the Constructive Cost Model

(COCOMO), we derived two different methods for recalibrating the model. Our conclusions for the multiple-input case were as follows:

- More accurate estimates can result from adding more information to the estimating relationship.
- All factors to be used in the estimating process should be used when deriving the estimating relationship.
- A cost estimating model calibrated to a specific environment yields more accurate estimates for that environment than one calibrated to a general environment.

The methods for estimating cost presented in this paper could improve the understanding and accuracy of software cost estimates for systems such as the Phase One SDS. The methods should be updated as more data on software of all types become available.

CONTENTS

FIGURES

TABLES

I. INTRODUCTION

Over the last 30 years, the use of software in the Department of Defense's weapon systems has grown tremendously. This growth has been not only in the size of the software programs used but also in the functions the software perform. Software in early weapon systems was used mainly for monitoring the condition of the hardware. Such a software program's size was less than 10 thousand lines of code (LOC). More recently, weapon system software has been used in the actual operation and control of the system, including such functions as signal processing, fire control, and command and control (C^2). Software size for these systems can be greater than 1 million LOC. Similar growth has been seen in the support software used for developing and maintaining the weapon system software. This tremendous growth in software size and function has made it difficult to accurately estimate the cost of developing software for a weapon system.

A number of methods have been devised for estimating software development costs.[1] These methods range from simple cost estimating relationships (CERs) to complex models. Despite the number of methods available, obtaining accurate estimates of software development costs remains a difficult task. The inaccuracy of the estimates resulting from these methods come from two sources: (1) inaccuracy of the input used and (2) inaccuracy of the estimating relationship employed. The work in this paper addresses the second source of inaccuracy.

The problem is not with the methods themselves, but with the way in which they are implemented. In order to use complex models, large amounts of information are needed, not only information about the software being estimated, but also about the model itself. Even the best cost model can give distorted estimates if it is used incorrectly.

A. OBJECTIVE

The objective of this work was to improve existing methods for estimating software costs. Our work concentrated on ways of implementing these methods. Both a simple CER and a complex model were examined.

---

[1] For a descriptive evaluation of software cost estimating tools, see Reference [1].

B. APPROACH

In order to come up with better ways to estimate cost, we examined the accuracy of estimates obtained from two sets of circumstances: the single-input case, in which a simple CER using the single input of size was used to estimate cost, and the multiple-input case, in which more information was added and a commercially available cost model called the Constructive Cost Model (COCOMO) [2] was used.

For the single-input case, we examined the data used to derive the CER to determine how to segregate the data so that the effect of non-size cost factors could be reduced. For the multiple-input case, we looked for ways to include factors other than size into the estimating relationship.

C. CONCLUSIONS

Two general conclusions can be drawn from our work:

- When using a single-input CER, predictive ability can be improved by segregating the data used to derive the CER into more homogeneous data sets.
- When using a commercially available cost model, predictive ability can be improved by recalibrating the model to a particular environment.

The sections that follow explain how we came to these conclusions.

II. BACKGROUND

The software for a complex system such as the Phase One SDS involves a wide variety of functions carried on several different platforms. The functions include signal processing, target discrimination, target tracking, fire control/weapons assignment, survivability, communications, operating systems, data gathering, command and control (C^2), and support functions. The software will be carried on space-, missile-, and ground-based platforms. Estimating the cost of such a complex system is a difficult task.

From the literature and from our discussions with organizations involved with software development and cost estimation, we can see that factors that influence cost are many. In the context of software cost estimation, cost factors come from three general environments:

- Software environment
  - Software size measured in source lines of code (SLOC) or function points
  - Software language
  - Software applications (function and platform)
- Development environment
  - Software engineering (tool availability, computer access, simulation capabilities, and configuration management, quality control, and requirements traceability)
  - Modern programming practices
  - Personnel (programmer and analyst) experience and capabilities
  - Requirements volatility
- Operating environment
  - Hardware architecture (types of processors, input/output requirements, and the number of devices in the network)
  - Time and memory constraints
  - Performance characteristics (the number and speed of targets to be tracked, reaction time, and the amount of data to be processed).

The factors that tend to influence cost the most are software size, software application and hardware architecture (i.e., the system's complexity), personnel experience and capabilities, and requirements volatility. One problem with incorporating these or other cost factors into CERs is that they are difficult to quantify so that they can be consistently applied.[2]

To illustrate the problem of consistent application, we use size as an example. Although size is the most widely accepted factor used in software CERs, how to measure it is subject to much debate. Should size be measured in lines of code (LOC), or should a measure of functionality, such as function points, be used?[3] Both measures need to be defined in a consistent manner to allow accurate interpretation of the measure as well as comparison across projects. Questions often arise about what constitutes a line of code, how LOC should be counted, and how to compare LOC for different languages. For example, should carriage returns, semi-colons, statements, comments, blank lines, and data declarations be included? The function point measure exhibits similar definitional problems.

For qualitative factors involving a rating, such as personnel capability, the consistency problem is even more apparent. If a rating system is used, the reliability of the measure is only as good as the judgment of those determining the rating. We have no way of knowing how the ratings compare across projects and contractors. Does the average capability of one contractor's personnel match that of another, having a similar impact on productivity and cost? Rating variables are often incorporated as multiplicative variables in an equation. In this form, the relative effects on cost are determined by expert opinion and limited statistical analysis. With these types of input, the question of how the judgment of the person doing the rating compares to expert opinion comes into play.

Another problem with incorporating cost factors into CERS has to do with availability of information. Estimates being made during the early development stages of a program are difficult simply because the information needed to make the estimates is not yet available. For example, software size is not known at the time of requirements definition. Often the only way to estimate LOC is either by expert opinion or by analogy to an

---

2 For a discussion of previous attempts to develop software CERs, see References [1] through [7].

3 For a more detailed discussion of function points and their use for real-time embedded software see References [6] and [7].

historical database. The detailed design information needed for function point estimates is not available at the time that such estimates are required.

Lack of knowledge similarly affects requirements volatility, the amount of major changes to requirements that occur over the life of a program. Because the number of requirements changes cannot be known until the program is complete, requirements volatility is a poor factor on which to base an estimate.

The dual problems of consistent quantification and lack of information have to do with the accuracy of the input to CERs. Once known, the inputs are used to estimate cost, but the problems do not end there. This paper addresses the problem of the inaccuracy of the methods used.

In general, CERs for software relate cost as a function of size. The relationship between size and cost is expressed in exponential form. Multiplicative factors can be introduced to capture the effects of factors other than size. The basic cost estimating relationship takes on the following form:

$$\text{Cost} = a(\text{Size})^b \prod_{i=1}^{n} F_i \ ,$$

where $\prod$ is the product of n multiplicatively introduced factors F. If the multiplicative factors are not included in the relationship, their influence on cost are only captured implicitly in the estimates of the parameters a and b.

In order to incorporate factors other than size into our software estimates, we segregated the data used to estimate parameters a and b so that the data would be as homogeneous as possible. Then we incorporated a set of factors representing additional cost drivers (as defined by the COCOMO cost estimating model) when estimating the parameters a and b. The data used to conduct this work is discussed in the next section.

III. DATA

The data used for our analysis came from the Jet Propulsion Laboratory (JPL) (provided by NASA) and from a report on military tactical aircraft development costs (MTADC) [8] (provided by the Air Force Electronic Systems Division (ESD), MITRE Corporation, and contractors). The JPL/NASA data were for software used in various satellite programs, including the Space Shuttle. Both ground- and space-based platforms were included. The MTADC data were on software whose functions included command, control, and communication (C^3), radar, and avionics integration on air-, sea-, and ground-based platforms. The data in both databases included information on size and cost

Software size was measured as total equivalent source lines of code, including data declarations, job-control language, and "include files," but excluding comments, prefaces, file-boundary statements, commercial off-the-shelf software, and non-delivered support and test software. In addition, software size was adjusted for whether the code was reused or adapted.

Software cost was measured in man-months of effort. This measure included management, design, programming, testing, and database administration. It did not include software installation. The duration of software development was measured from completion of the system design review to completion of the functional qualification test, which takes place before system integration.

Limiting factors in our analysis were the lack of historical data and the lack of knowledge about the data that were available. Although 101 observations were collected, knowledge of individual data points was limited. For some data, only the platform of the software was known with certainty, and the functions were known only at a general level. For other data, we only knew in general terms what functions and platforms were involved. For the JPL/NASA data we were able to accurately distinguish between ground- and space-based data, but because software functions were not completely identified, further segregation on that basis was not possible. Similarly, for the MTADC data, neither function nor platform were known with enough confidence to segregate within the database.

Despite these limitations, some segregation was done with significant implications. The JPL/NASA data involved space-oriented software. That is, either the software was located in space or was ground-based software that supported a space mission. The type of functions involved were operational and system, but could not be distinguished given the limited knowledge of the data. Using these data, we were able to test for the difference between software located in space and software located on the ground.

The MTADC database involved air-, sea-, and ground-based software, but they were not distinguished. We knew that the functions involved were radar, C^3, and avionics integration activities. With these data, we tested for the difference between what we call avionic (embedded real-time) functions and space functions.

The data were thus segregated into three separate databases: JPL/NASA space data, JPL/NASA ground data, and MTADC avionics data. An analysis was performed to determine, first, if the three segregated databases were significantly different from each other, and if so, if three separate equations would produce better results than one. Second, we incorporated other, non-size cost factors into the basic equation by recalibrating the COCOMO model. This work is described in the next section.

IV. ANALYSES

In this section, we address two hypotheses concerning the accuracy of relationships used to estimate software development cost. The first hypothesis, addressed by the single-input case, is that a relationship based on data tailored to a specific environment yields more accurate estimates for that environment than a relationship based on a general environment. The implication is that when estimating cost for a software project consisting of various combinations of functions and platforms, more than one equation should be used.

The second hypothesis, addressed by the multiple-input case, is that when adding factors to the estimating relationship for predictive purposes, the factors should be included in the analysis to derive the estimating relationship. Building on the first hypothesis, the implication is that when constructing a relationship based on data tailored to an environment with additional, non-size factors included, the basic relationship between size and cost should be re-derived using those non-size factors.

A. SINGLE-INPUT CASE

The single-input method used the cost factors of size and application (function and platform) to estimate cost. That is, we segregated the database according to the application of the software and then estimated cost as a function of size. The purpose of segregating the database was to reduce the variance caused by non-size factors that influence cost. The application of the software may be correlated with the non-size factors that are difficult to quantify. As an example, software projects that operate in space are all subject to similar hardware architecture constraints. By deriving a CER for space-based software only, we eliminate the need to quantify the effect of hardware architecture on cost.

1. Size and Application

To ensure consistency of the size measure across projects in the analysis, size was measured as total equivalent source lines of code (LOC). In addition, software size was adjusted for whether the code was reused or adapted.

The application of software refers to both the platform and function of the software. Classes of software functions include operational, system, and support, as follows:

- Operational: performs prime mission, e.g., target tracking, command and control, and fire control
- System: infrastructure to other functions, e.g., operating system, security, communications, and fault tolerance
- Support: off-line software, e.g., simulation/training, maintenance/site support, and report generators.

We concentrated first on examining the available data to determine possible methods of segregation. The analysis was performed to demonstrate that for the different applications, the relationship between size and cost is significantly different.

We expected a difference in productivity, as measured in lines of code per man-month (LOC/MM) or hours per lines of code (HOURS/LOC), both between the various platforms and between the various functions. Examining the platforms first, we expected ground- and sea-based software to exhibit the highest productivity, air-based software the next highest, and space-based software the lowest.

The non-size factors that influence cost that tend to be similar by platform are hardware environment factors and quality-control factors. The hardware environment factors include the architecture (processors, communication busses, etc.) and time and memory constraints. The more complex the architecture, the more difficult and less productive the software development. Similarly, the more time and memory constraints, the more difficult and less productive the software development. We expected architecture complexity and time and memory constraints to be greater in space than in air. Similarly, they should be greater in the air than on ground or sea, where they should be about the same. This led us to the conclusion that space-based software exhibits less productivity then air-based software, which exhibits less productivity than ground- or sea-based software.

The quality-control factors to be considered are configuration management, testing requirements, and documentation (or traceability) requirements. Generally, the easier it is to service the software once operational, the more lax are the quality-control efforts. Simply put, the access to space-based software is difficult at best; therefore, space-based software should experience greater configuration management and testing requirements during development then air-based software. Similarly, air-based software should have

greater requirements then ground- or sea-based software, which should be about the same. The greater requirements should lower productivity; that is, more effort per LOC is expected, thus lowering the number of LOC/MM.

In terms of the various functions and their relationship to productivity, the main factor involved is processing complexity. Processing complexity is basically the difficulty level of the algorithms and logic involved, including the mathematical applications. Simply put, system code is expected to be the most complex, followed first by operational code and then by support code.

Our approach assumed that non-size factors that affect cost tend to be similar for a given software application, so the variance caused by these factors are reduced significantly by segregating the software according to application. Our effort attempted to demonstrate that the data can be segregated, improving predictive ability, and to reveal the relative differences between applications. Segregating the data avoids the problem of quantifying the variables that are difficult to measure consistently.

There are, of course, other factors that influence cost that apply across application types. We acknowledge that segregating by application does not capture these effects, and that the problem of quantifying these measures still exists.

2. Modeling

The CERs we derived estimate cost as a function of size. The relationship between size and cost is an exponential function of the basic form, cost = $a(size)^b$. The coefficients were estimated using a logarithmic transformation of the basic form, yielding the following:

$$\ln(cost) = \ln(a) + b[\ln(size)]. \quad (1)$$

A Chow test[4] was performed to verify that a pair-wise combination of sets of data within a larger group of data sets should be segregated. For data sets h and g with m and n degrees of freedom, respectively, and k parameters to be estimated, the Chow test for segregating data involved five basic steps:

1. Use Equation (1) to estimate cost using data sets h and g separately.
2. Use Equation (1) to estimate cost using the combined data sets h and g.

---

[4] For a more detailed discussion of the Chow test, the reader is referred to Reference [9].

3. Perform the following calculation

$$F = \frac{(RSS_{h+g} - RSS_h - RSS_g)/(K + 1)}{(RSS_h + RSS_g)/(M + N - 2(K + 1))},$$

where RSS is residuals squared sum.

4. State the null hypothesis: data sets h and g should be pooled.

5. If the F value calculated in step 3 is greater then the critical value of F (CV) with (K + 1, M + N - 2(K + 1)) degrees of freedom, then reject the null hypothesis.

If the result of the Chow test is to reject the null hypothesis, then the two data sets should not be pooled. The implication of that Chow test result is that individual equations estimated for the segregated databases would be more accurate predictors then one equation for the combined database.

To further demonstrate the improved predictive ability from segregating the data, the predicted values were compared with the actual values. That is, the size values were input to the resulting equations with the estimated effort compared to the actual effort. The equations were compared on the basis of how often the predicted value fell within 20 percent of the actuals.

Finally, an attempt was made to demonstrate the relative differences between the various applications. This was done by inputting a range of sample size values (10 thousand to 1000 thousand LOC). The predicted cost outputs (in MM) were compared after transforming them into productivity measures (in HOURS/LOC). The comparison was made for a given size and for increasing size.

B. MULTIPLE-INPUT CASE

1. Adding Cost Factors

The multiple-input approach attempted to explicitly incorporate additional cost factors into the equation. That is, we estimated the coefficients a and b with non-size cost factors in the estimating equation such that a and b no longer implicitly include their effects. The solution to the problem of measuring the factors was to use subjective variables as defined in the Constructive Cost Model (COCOMO) developed by Boehm [2]. These variables, known as Effort Adjustment Factors (EAFs), consist of fifteen factors divided into four groups—product, project, development, and computer attributes—which

correspond to the three classes of cost factors previously discussed. Although not complete, they provide adequate representation and are available. The EAFs are discrete variables in that they involve a rating scheme (e.g., very low, low, nominal, high, very high) and each rating has a discrete numerical value. The EAFs are presented in Table 1.

Table 1. Software Development Effort Multipliers

	Ratings					
Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY (Required Software Reliability)	.75	.88	1.00	1.15	1.40	
DATA (Data Base Size)		.94	1.00	1.08	1.16	
CPLX (Product Complexity)	.70	.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME (Execution Time Constraint)			1.00	1.11	1.20	1.66
STOR (Main Storage Constraint			1.00	1.06	1.21	1.56
VIRT (Virtual Machine Volatility)[a]		.87	1.00	1.15	1.30	
TURN (Computer Tunaround Time)		.87	1.00	1.07	1.15	
Personnel Attributes						
ACAP (Analyst Capability)	1.46	1.19	1.00	.86	.71	
AEXP (Applications Experience)	1.29	1.13	1.00	.91	.82	
PCAP (Programmer Capability)	1.42	1.17	1.00	.86	.70	
VEXP (Virtual Machine Experience)[a]	1.21	1.10	1.00	.90		
LEXP (Programming Language Experience)	1.14	1.07	1.00	.95		
Project Attributes						
MODP (Use of Modern Programming Practices)	1.24	1.10	1.00	.91	.82	
TOOL (Use of Software Tools)	1.24	1.10	1.00	.91	.83	
SCED (Required Development Schedule	1.23	1.08	1.00	1.04	1.10	

Source: Reference [2], p. 118.

[a] For a given software product, the underlying virtual machine is the complex of hardware and software (OS, DBMS, etc.) it calls on to accomplish its tasks.

COCOMO comes in three versions, basic, intermediate, and detailed. The difference among the three versions is the degree of information input into the model. Each version of the model has three modes: embedded, semi-detached, and organic. The

differences among the modes is the type of application of the project, as reflected in different values for a and b. We used the embedded mode of the intermediate version of the model.

The algorithm of the intermediate model is of the basic form already described. That is, cost is first estimated as an exponential function of size and then adjusted by the EAFs. The form of the equation is:

$$\text{COST} = a(\text{SIZE})^b \prod_{i=1}^{15} \text{EAF}_i \quad .$$

Boehm used data from TRW to derive values for a and b. These values were derived taking the EAFs into account explicitly by adjusting cost by the product of the EAFs. The discrete numerical values for the individual EAFs were derived heuristically using expert opinion at TRW.

Because the equation is a representation of the software development process at TRW, it should be recalibrated to the environment where the model is used. The approaches described are different methods for recalibrating COCOMO.

The most commonly recalibrated parameters are a and b. There are two methods for deriving new values for a and b: (1) without the EAFs and (2) with the EAFs. The first method is the single-input approach described in the previous subsection. The second method is the multiple-input approach described here.

Two concerns arise when recalibrating with subjective variables like the EAFs. These concerns are: (1) the accuracy and consistency with which the EAFs are assessed for the given projects and (2) the validity of the numerical assignment given to the specific ratings for each EAF. For the most part, we are dependent upon the organizations that develop the software project and put together the database to maintain accuracy and consistency in assessing ratings. Ideally, one person who is intimately knowledgeable about all the projects within a database should maintain consistency and objectivity in assessing the ratings.

The numerical values for the EAFs should be derived statistically and for a particular environment. Attempts have been made at this but the process proves difficult to obtain good results.[5] Briefly, this has involved incorporating, for each EAF, a dummy

---

[5] For examples, the reader is referred to [10] and [11].

variable for each rating but one (say, nominal, assuming it equals one) on the right-hand side of the equation. This would potentially involve over 50 independent variables in the regression. Unfortunately, this would both create a cumbersome equation and require a large number of observations.

In the single-input approach we demonstrated the desirability of segregating the data. Building on this, the multiple-input approach recalibrates COCOMO using the segregated data. We attempted to show that the EAFs can be used to derive new values for a and b, resulting in a more accurate prediction of cost.

2. Modeling

The multiple-input approach uses two different methods to incorporate the EAFs into the model. One method involves incorporating them as a single product, the other as four individual products, one for each of the four groups—product, project, development, and computer.

By incorporating the EAFs explicitly into the estimating equation, the coefficients in the basic equation a and b no longer implicitly reflect the effects of the EAFs on cost. The result from incorporating the EAFs should be that the difference in the coefficients a and b in the separate equations is reduced. That is, we would expect a and b to take on more similar values when incorporating the non-size factors into the equation.

The model for the first method is of the exponential form as follows:

$$\text{COST} = a(\text{SIZE})^b * \prod_{i=1}^{15} \text{EAF}_i \quad .$$

The equation is first transformed as follows:

$$\text{COST}/\prod_{i=1}^{15} \text{EAF}_i = a(\text{SIZE})^b \quad .$$

The parameters (a and b) are then estimated using a logarithmic transformation, yielding the following:

$$\ln\left(\text{COST}/\prod_{i=1}^{15} \text{EAF}_i\right) = \ln(a) + b\,(\ln(\text{SIZE})) \quad .$$

For this method the data was again segregated.

The model for the second method was also of the exponential form:

$$COST = a(SIZE)^{b} * \Pi_1^{c_1} * \Pi_2^{c_2} * \Pi_3^{c_3} * \Pi_4^{c_4} ,$$

where

Π_1 = RELY * DATA * CPLX

Π_2 = TIME * STOR * VIRT * TURN

Π_3 = ACAP * PCAP * AEXP * LEXP * VEXP

Π_4 = MODP * TOOL * SCED.

The coefficients to be estimated were a, b, c_1, c_2, c_3, and c_4. The coefficients c_1, c_2, c_3, and c_4 were used to relate the impact of the individual EAFs on cost more directly.

To estimate all the coefficients, a logarithmic transformation was used, yielding the following:

$$\ln (COST) = \ln (a) + b [\ln(SIZE)] + c_1 [\ln(\Pi_1)] + c_2 [\ln (\Pi_2)] + c_3 [\ln(\Pi_3)] + c_4 [\ln(\Pi_4)] .$$

For this method, we again attempted to segregate the data when estimating the coefficients.

Finally, we demonstrated which method produces the best predictive equation by determining how often the predicted falls within 20 percent of the actual. The methods just described were also compared to Boehm's original equation and the single-input equations.

V. RESULTS

A. SINGLE-INPUT CASE

Table 2 presents the regression results for the combined data and the individual data sets. The results are reported after transforming the estimated logarithimic equation back to a multiplicative form. The transformations introduce a bias in the constant term. We adjusted for this bias by adding one-half of the regression mean square error to the constant term of the logarithmic equation. This adjustment is an approximate correction for the bias introduced by the logarithmic estimation method.

Table 2. Regression Results for the Single-Input Case

Data	Cost Estimating Relationship	N	R^2	Adj. R^2	SEE
Combined	$MM_C = 9.36\ (KLOC)^{1.00}$ (8.44) (17.04)	101	.75	.74	.78
JPL/NASA Ground	$MM_G = 4.18\ (KLOC)^{1.05}$ (9.3) (28.7)	69	.92	.92	.40
MTADC Avionics	$MM_A = 5.8\ (KLOC)^{1.21}$ (2.79) (8.45)	18	.81	.80	.67
JPL/NASA Space	$MM_S = 30.72\ (KLOC)^{1.009}$ (9.6) (10.5)	14	.90	.89	.55

Note: The numbers in parenthesis are the t-statistics for the regression coefficients.

Figure 1 presents the three individual equations in graphical form. Note that for only the MTADC avionics data does the exponent take on a value similar to previous efforts (see References [2] through [4]). The exponent value is greater then one, suggesting diseconomies of scale. For the JPL/NASA data, the exponents take on a value of approximately one.

The question of whether the differences between coefficients in the combined equations and in the individual equations was examined in two ways, statistically through a Chow test and graphically through a display of predictive accuracy. The Chow test

determines if the differences are statistically significant and, if so, that the data should be segregated. The graphic presentations demonstrate the predictive differences.

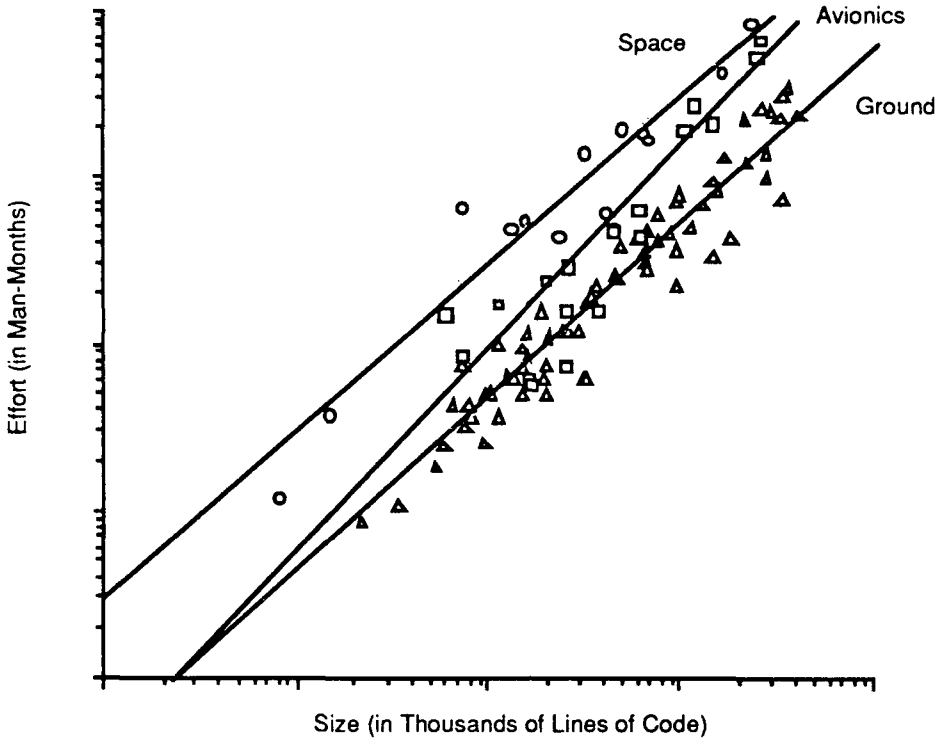

Figure 1. Effort Versus Size by Platform for the Single-Input Case

Table 3 presents the results of the Chow test. Since the calculated F valve (F) is greater then the critical valve of F (CV), the Chow test supports the expectation that the individual databases are significantly different. The conclusions from these tests are that the data should be segregated and individual CERs derived.

Table 3. Chow Test and Mean Squared Error Test Results

Chow Test Applications	F	CV (0.1 level)
JPL/NASA Ground		
Versus Space	96.1	3.1
Versus MTADC Avionics	20.0	3.1
JPL/NASA Space		
Versus MTADC Avionics	11.2	3.3

Figure 2 presents the results of determining the frequency with which the predicted fall within 20 percent of the actuals. The implication of these results is that an equation derived from a segregated data set is more accurate than an equation derived from a more general data set.

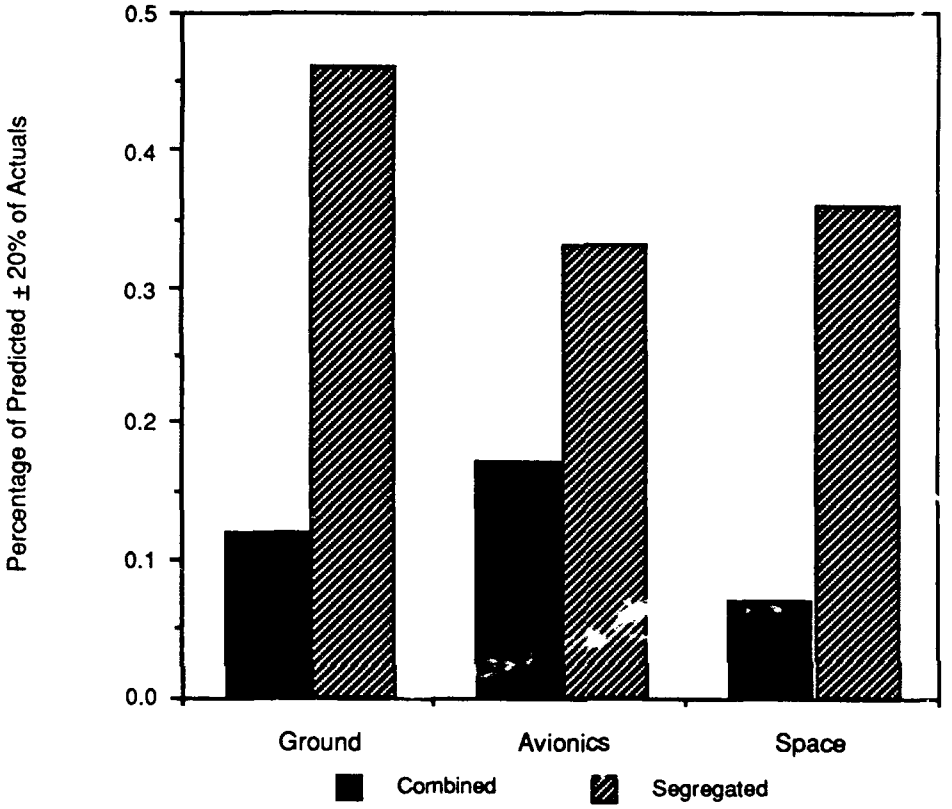

Figure 2. Comparison of Estimating Equations by Database for the Single-Input Case

Table 4 presents the results of using a range of sample input to generate sample output for the various equations. The results are transformed into an HOUR/LOC measure (this assumes a 152-hour man-month).

The conclusions drawn are that space-based software is approximately six times as expensive to develop as ground-based software, given size. For the MTADC avionics data, the conclusion is that, depending on the size of the input, avionics software can be from two to four times as expensive to develop as ground-based space application software. What is interesting to note, again, is that only for the avionics software does productivity vary significantly with the size of the project.

Table 4. Comparison of Productivity

Software Applications	Size Input (Thousands of Lines of Code) 10	50	100	500	1,000
Space	4.8	4.8	4.9	4.9	5.0
	(6.9)	(6.2)	(6.1)	(5.6)	(5.5)
Avionics	1.4	2.0	2.3	3.3	3.8
	(2.0)	(2.5)	(2.9)	(3.8)	(4.2)
Ground	0.7	0.8	0.8	0.9	.9

Note: The numbers in parentheses are the ratios of the applications to the ground-based data, used as a base case.

B. MULTIPLE-INPUT CASE

1. EAFs as a Single Product

Table 5 presents the regression results for incorporating the EAFs as a single product. Three equations, one for each segregated data set, are shown. Figure 3 presents the results in graphical form. As can be seen from both the regression results and the graph, the estimated parameters are more similar when the EAFs are included than when they are not (see the previous subsection).

Table 5. Regression Results for the Multiple-Input Case

Platform	Cost Estimating Relationship	N	R^2	Adj. R^2	SEE
Ground	$MM = 4.08\ (KLOC)^{1.09}$ (12.43) (39.43)	69	.96	.96	.30
Avionics	$MM = 5.85\ (KLOC)^{1.1}$ (2.3) (6.45)	16	.75	.73	.76
Space	$MM = 8.23\ (KLOC)^{1.08}$ (7.4) (14.1)	14	.94	.94	.44

Note: The numbers in parenthesis are the t-statistics for the regression coefficients.

Again, a Chow test was performed to verify that, with the EAFs included, the data should be segregated. Although not reported here, the results of the Chow test supported segregating the data. No unambiguous pooling of the data could be performed. The fact

that the data can be segregated with the EAFs included implies that factors other than the EAFs vary by platform.

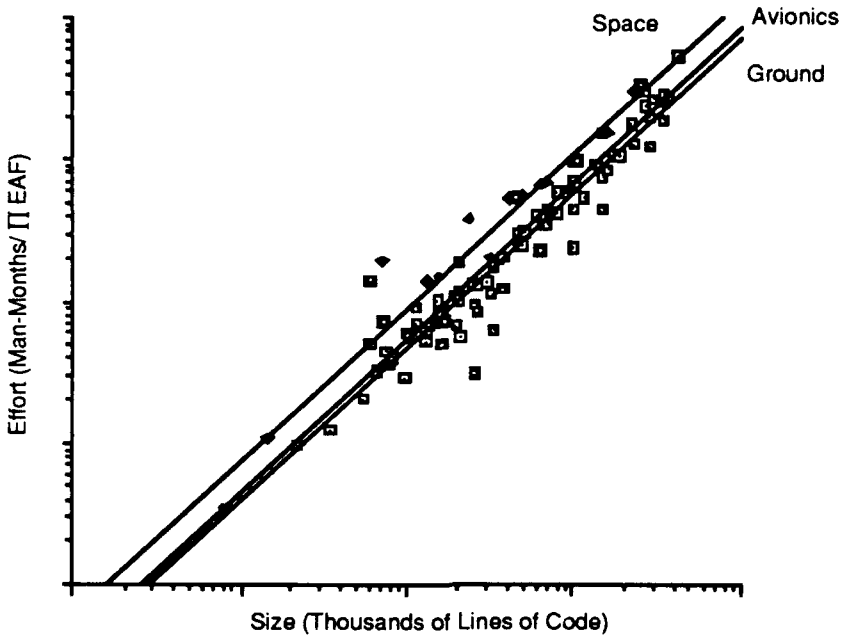

Figure 3. Effort Versus Size by Platform for the Multiple-Input Case

Figure 4 presents the results of determining the frequency that the predicted is within 20 percent of the actuals. Table 6 presents the estimating methods compared in the figure. The Boehm equation is from the uncalibrated intermediate COCOMO model. The implication from these results is that, except for the avionics, adding the EAFs improves the predictive ability (single input vs. multiple input). For all three databases, if the EAFs are used for prediction, there is improvement when the EAFs are used to estimate the parameters a and b (single input plus vs. multiple input). For all three databases, predictive ability is improved when the model is recalibrated (multiple input vs. unrecalibrated).

Table 7 presents the mean values for the product of the EAFs by database. For both single-input methods, the parameters a and b were derived without the EAFs. Only in the case of the ground data did predictive ability improve when using the EAFs to predict. For both the space and avionics data, the predictive ability decreased. The difference is due to the value of the EAFs. When the product of the EAFs is not significantly different from one, no loss of predictive ability occurs. Only when the product of the EAFs differ significantly from one is there a loss in predictive ability.

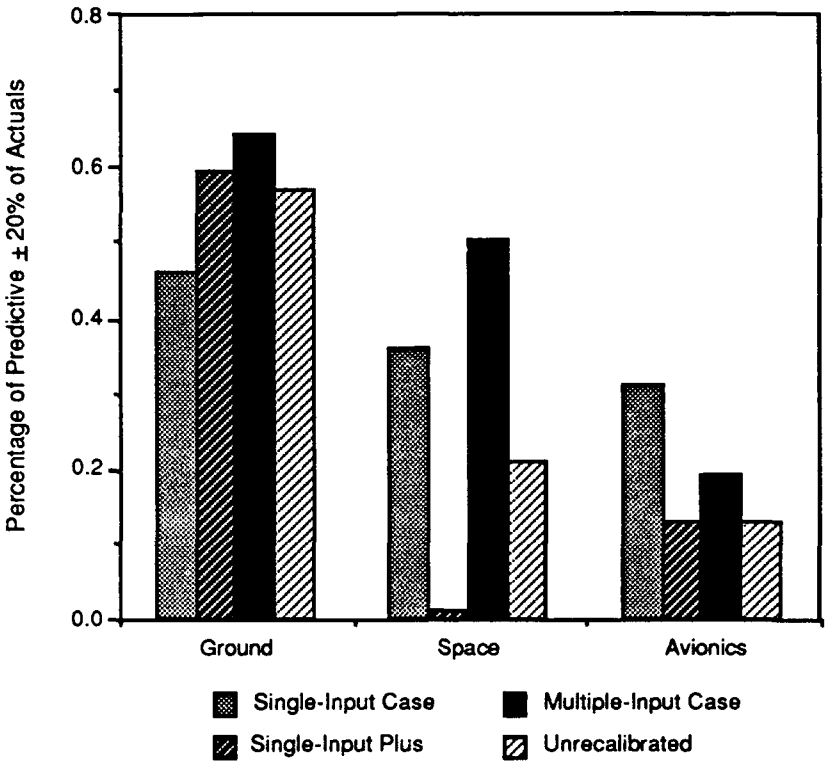

Figure 4. Comparison of Estimating Methods by Database for the Multiple-Input Case

Table 6. Comparison of Estimating Methods

Method name	Parameters Estimation Method	Prediction Method
Single Input	$MM = a(KLOC)^b$	$MM = a(KLOC)^b$
Single Input Plus	$MM = a(KLOC)^b$	$MM = a(KLOC)^b \prod_{i=1}^{15} EAF_i$
Multiple Input	$\frac{MM}{\prod_{i=1}^{15} EAF_i} = a(KLOC)^b$	$MM = a(KLOC)^b \prod_{i=1}^{15} EAF_i$
Unrecalibrated	Not Applicable[a]	$MM = 2.8(KLOC)^{1.2} \prod_{i=1}^{15} EAF^i$

[a] Taken from Reference [2].

Table 7. Mean Values for the Product of the EAFs

Database	Mean $\prod_{i=1}^{15} EAF_i$
Ground	.921
Avionics	1.77
Space	3.05

2. EAFs as the Product of Four Groups

Our data were insufficient to completely test the case where the EAFs are included individually as products of the four groups. Results were obtained when the space and ground data were pooled and a dummy variable used to differentiate between the two classes of data. The dummy variable DSPACE was used to differentiate between space and ground data, where DSPACE has a value of one if the project is space based, a value of zero otherwise.[6] The regression results are:

$$MM = \underset{(11.82)}{(3.96} * \underset{(5.13)}{2.38^{(DSPACE)}})(KLOC)^{\underset{(35.83)}{1.07}} * \Pi_1^{\underset{(4.58)}{1.05}} * \Pi_2^{\underset{(4.43)}{.7}} * \Pi_3^{\underset{(4.43)}{.72}} * \Pi_4^{\underset{(2.4)}{.77}}$$

$$N = 83 \qquad \text{Adj. } R^2 = .96^2 \qquad SEE = .1$$

When the predicted values were compared to the actual values, no improvement was found relative to including the EAFs as a single product. Although no results are reported here, this method shows promise. When more data become available, further work along these lines is warranted.

C. CONCLUSIONS

The conclusions to be drawn from the single-input analysis are listed below:

- The effect on cost of factors that are difficult to quantify can be captured by segregating the data according to software application, resulting in more accurate predictive ability.
- Software developed for space is six times more expensive than software developed for ground application, given size.

[6] This is not consistent with running individual regressions reported earlier.

- Software developed for avionic-type applications (C^3, radar, and integration software residing in air, on sea, or on ground) is two to four times as expensive as software developed to support space activities residing on the ground. The range depends on the size of the project.

The conclusions to be drawn from the multiple-input analysis are listed below:

- Adding more information to the estimating relationship can improve the predictive ability.
- If additional factors are to be used in the estimating process, they should be used to generate the estimating relationship.
- A recalibrated cost estimating model for a specific environment will predict more accurately than a cost estimating model calibrated to a more general environment.

The single-input analysis suggests the need for more homogeneous data in the derivation of a cost estimating relationship. The multiple-input analysis, suggests that inclusion of more information in the derivation of the relationship and then in estimating can further improve the predictive ability.

VI. EXAMPLE APPLICATION

In this section, we present an example of how the different equations presented in this paper are used. The example shows both the single-input case and the multiple-input case. The steps taken to generate a development cost estimate are: (1) identify the types of software involved, (2) identify the source lines of code and effort adjustment factor (EAF) ratings for each type of software, (3) choose the correct equation for each type of software, and (4) perform the appropriate calculations to generate estimates.

The system being estimated is a fictitious reconnaissance satellite system designated R-1. The software used in R-1 exists both in space and on the ground. The functions located in space include signal and data processing for an infrared sensor, communications and system software, and software associated with flying and maintaining the satellite in space. The functions located on the ground include command and control, duplication of the signal and data processing done in space, communication, monitoring and testing of the software and hardware in space, and data gathering, report generating, and similar off-line functions.

The first step to generating an estimate is to identify the types of software in order to place them in categories for cost estimating purposes. For the R-1 example, three types of categories of software are identified. All software operating in space is placed in one category known as space software. The real-time embedded functions on the ground that are similar to command and control and radar applications are placed in the category known as avionics. Even though these functions do not exist in air, they are similar to avionic-type functions. The monitoring and test software and the off-line software are placed in the category known as ground software.

The second step is to identify the source lines of code (LOC) and EAF ratings for each type of software. The R-1 satellite system has a total of 1 million LOC with 250 thousand LOC as space, 300 thousand as avionics, and 450 thousand as ground. Table 8 presents the EAFs by type for R-1, along with the size figures. Two sets of EAFs for each type of software are presented. Set 1 represents EAFs that are similar to the data used to generate the estimating relationships. Set 2 represents EAFs that differ from the data used to generate the estimating relationships.

The third step is to choose the correct equation for each type of software. In our example the software was divided into three categories that matched the equations derived in Section IV. That is, we used the space equation for the space software, the avionics equation for the avionics software, and the ground equation for the ground software.

Table 8. Program Characteristics for the R-1 Satellite System

	Software Category					
	Ground		Avionics		Space	
Characteristic	Set 1	Set 2	Set 1	Set 2	Set 1	Set 2
Size (thousands of LOC)	450	450	300	300	250	250
EAFs						
RELY	High	Very High	High	Very High	Very High	High
DATA	Nominal	Very High	High	High	Nominal	Nominal
CPLX	High	Very High	Very High	Extremely High	Very High	Extremely High
TIME	Nominal	Nominal	Very High	Very High	Very High	Very High
STOR	Nominal	Nominal	Very High	Very High	Extremely High	Extremely High
VIRT	Low	Nominal	Low	High	Nominal	High
TURN	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal
ACAP	High	Nominal	High	Nominal	High	Nominal
AEXP	High	Nominal	High	Nominal	High	Nominal
PCAP	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal
VEXP	Low	Low	Low	Low	Low	Low
LEXP	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal
MODP	High	Nominal	High	Nominal	High	Nominal
TOOL	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal
SCED	Nominal	Nominal	Nominal	Nominal	Nominal	Nominal

The fourth step is the actual execution of the equations with the appropriate inputs. Two assumptions were made in order to generate cost estimates. One was that the software can be partitioned into units of 100 thousand LOC; therefore, 100 thousand LOC is the input to the equation with the result multiplied by the actual size divided by 100 thousand LOC. As an example, for the 250 thousand LOC of space software, 100 thousand LOC is input to the equation and the result is multiplied by 2.5. Since the equations yield results in

man-months (MM), a conversion is made to dollars by assuming a rate of $12,500/MM, which is equivalent to an annual rate of $150,000.

Table 9 presents the results for the single-input case, and Table 10 shows them for the multiple-input case.

Table 9. Equations and Results for the Single-Input Case

Software Category	Equation	Man-Months of Effort	Cost in Millions of Dollars
Ground	$4.18\ (KLOC)^{1.05}$	2,368	$30
Avionics	$5.8\ (KLOC)^{1.21}$	4,577	$57
Space	$30.72\ (KLOC)^{1.009}$	8,005	$100
Total		14,950	$187

Table 10. Equations and Results for the Multiple-Input Case

Software Category	Equation	$\prod_{i=1}^{15} EAF_i$	Man-Months of Effort	Cost in Millions of Dollars
Ground	$4.08\ (KLOC)^{1.09} \prod_{i=1}^{15} EAF_i$	Set 1: 0.97 Set 2: 2.32	Set 1: 2,696 Set 2: 6,453	Set 1: $34 Set 2: $81
Avionics	$5.85\ (KLOC)^{1.1} \prod_{i=1}^{15} EAF_i$	Set 1: 1.73 Set 2: 4.58	Set 1: 4,815 Set 2: 12,746	Set 1: $60 Set 2: $159
Space	$8.23\ (KLOC)^{1.08} \prod_{i=1}^{15} EAF_i$	Set 1: 2.89 Set 2: 5.91	Set 1: 8,595 Set 2: 17,570	Set 1: $107 Set 2: $220
Total			Set 1: 16,102 Set 2: 36,769	Set 1: $201 Set 2: $460

An important question to ask when estimating software development cost is which set of equations to use, the single-input or the multiple-input case. Using the EAFs in the multiple-input case requires more information then the single-input case. Will this information necessarily improve the accuracy of the estimate?

In the example application two sets of EAFs for each type of software are provided. Set 1 approximates the mean of our data, and Set 2 differs from the mean but is still within the range of our data. Examining Set 1 first, we see that the cost estimate for the multiple-input case differs from the single-input case by approximately 7%, not a significant difference. Therefore, the EAFs did not add significantly to the estimating process. In Set 2, we see that the cost estimate for the multiple-input case differs from the single-input case by about 145%, a significant difference. In this instance the EAFs do add significantly to the estimating process. The EAFs can thus be used to fine-tune an estimate when software being estimated is different from the data used to derive the estimating relationship.

An additional concern is the confidence the user has in the assessment of the EAFs. Early in the life cycle of a project, the EAFs may not be known with confidence. Inaccurately assessed EAFs can affect the accuracy of the estimate. Similarly, inconsistent assessment of EAF values across components of a project can distort estimating ability.

VII. SUMMARY AND CONCLUSIONS

This paper details two separate attempts to improve the accuracy of the relationships used in estimating software development cost. The single-input analysis is representative of using a simple CER with only one input (size). The multiple-input analysis is representative of using a commercially available model, namely COCOMO.

The intent of the single-input analysis was to show that one CER was less accurate than several CERs considering the variety of software involved in systems such as the Phase One SDS. The basic approach was to segregate historical data according to platform and application, derive a CER for each segregated database, and then show the individual CERs to be more accurate than a combined CER.

The conclusions from this effort were:

- The effect that factors that are difficult to quantify have on cost can be captured by segregating the data according to software application, resulting in more accurate predictive ability.
- Software developed for space is six times more expensive than software developed for ground application, given size.
- Software developed for avionic-type applications (C^3, radar, and integration software residing in air, on sea, or on ground) is two to four times as expensive as software developed to support space activities residing on the ground. The range depends on the size of the project (number of LOC).

The intent of the multiple-input analysis was to show that predictive ability could be increased by adding information. The result of this effort was to provide a method for recalibrating COCOMO. Two different methods were derived for recalibrating COCOMO, then several methods were compared to determine how best to improve predictive ability.

The conclusions for this effort were:

- Adding more information to the estimating relationship can improve the predictive ability.
- If additional factors are to be used in the estimating process, those same factors should be used when generating the estimating relationship.

- A recalibrated cost estimating model for a specific environment will predict more accurately than a cost estimating model calibrated to a more general environment.

To summarize, methods were developed for estimating cost in this effort. If adopted, these methods could improve the understanding and accuracy of the software cost estimates for Phase One SDS. The cost estimating methods should be continually updated as new data becomes available.

REFERENCES

[1] Bailey, Elizabeth K., Thomas P. Frazier, and John W. Bailey. *A Descriptive Evaluation of Automated Software Cost-Estimation Models.* IDA Paper P-1979, Institute for Defense Analyses, October 1986.

[2] Boehm, Barry W. *Software Engineering Economics.* Englewood Cliffs, NJ; Prentice-Hall, Inc. 1981.

[3] Frederic, Brad. *A Professional Model for Estimating Computer Program Development Cost.* TM-7, Tecolote Research, Inc., December 1974.

[4] Funch, Paul G. *Software Cost Database.* MTR 10329, MITRE Corporation, October 1987.

[5] Habib-Agahi, Hamid, James Quirk, and Shantanu Malhotra. *Software Productivity and Costs in NASA Projects.* JPL 900-990, Jet Propulsion Laboratory, January 1988, Draft.

[6] Reifer, Donald J. ASSET-R: *A Function Point Sizing Tool For Scientific and Real-Time Systems.* RCI-TN-299, Reifer Consultants Inc., September 1987.

[7] Tom, Kenneth B., and Everett E. Ayers. *Software Sizing and Cost Estimating.* ARINC Research Corporation, July 1985.

[8] Harmon, Bruce R., et al. *Military Tactical Aircraft Development Costs, Vol. I, Summary Report.* IDA Report R-339, Institute for Defense Analyses, September 1988.

[9] Pindyck, Robert S. and Daniel L. Rubinfeld. *Econometric Models and Economic Forecast,* 2nd edition, New York: McGraw Hill Book Company, 1981.

[10] Gulezian, Ronald. *Calibrating and Transforming COCOMO,* Mimeograph.

[11] Brenner, Neal J. *A Statistical Analysis of the COCOMO Database.* TR-0015/1, Tecolote Research Inc., October 1989, Draft.

ABBREVIATIONS

C^2	command and control
C^3	command, control, and communications
CER	cost estimating relationship
COCOMO	Constructive Cost Model
EAF	Effort Adjustment Factor
ESD	Electronic Systems Division
JPL	Jet Propulsion Laboratory
KLOC	thousands of lines of code
LOC	lines of code
MM	man-months
MTADC	military tactical aircraft development costs
NASA	National Aeronautics and Space Administration
POET	Phase One Engineering Team
SDIO	Strategic Defense Initiative Organization
SDS	Strategic Defense System