The Comparison and Improvement of Effort Estimates from Three Software Cost Models

Dennis Kafura
Sallie Henry
Mark Gintner

TR 87-37

The Comparison and Improvement of Effort Estimates from Three Software Cost Models

by

Dennis Kafura,
Sallie Henry
and
Mark Gintner

Computer Science Department
Virginia Tech
Blacksburg, VA 24061

Abstract

This paper presents an empirical investigation of effort estimation techniques using three software cost models: Boehm's basic COCOMO model, Thebaut's COPMO model, and Putnam's model. The results, based on proprietary historical project data provided by a major computer manufacturer, are in three parts: (1) a comparison of the three models showing that more accurate estimates are obtained from the COPMO model; (2) improvements in the COPMO estimates by means of two techniques termed "submodeling" and "constructive modeling"; and (3) the definition and evaluation of a promising new technique, termed "adjustment multipliers", for improving a model's estimates. Finally, a second set of industrial data is used to confirm the general applicability of these improvement techniques.

I. INTRODUCTION

Software cost models continue to be of interest to both the research and business communities. Researchers use cost models as a means to understand and explain the dynamics of the software development process. More pragmatically, the business community looks on these models as a device for estimating, and ultimately controlling, the cost of software projects which have experienced overruns approaching two hundred percent [DeM82]. While referring to these models as "cost" models they are, more specifically, models for estimating the total effort (measured in person-months) required to produce a software product. Estimating effort is a common goal of many cost models since effort in a key factor in the overall dollar costs of software production.

In this paper we report on the results of a collaborative study involving both the research and industrial communities. The first goal of the study was to perform an empirical comparative evaluation of three cost models against historical data collected from completed commercially developed software projects. This part of the study, similar to those conducted by Kemerer [Kem87] and Thebaut [The83], is present in the first part of Section IV. The second goal was to improve the estimation accuracy of one of these models specifically for our industrial partner's environment. In the second part of Section IV we present the two techniques which we used and which we believe can be used not only in other environments but also with models different from the one which we chose. One technique, termed submodeling, employs a cost model which is calibrated differently for different subsets of projects. A second technique, termed constructive modeling, produces an overall system cost estimate by summing the estimates of individual subsystems. While these two general techniques have been used previously, we believe that we have applied them in a different manner. The third and most novel goal of the study was to explore a new and highly promising cost estimation method. This new technique, which we call adjustment multipliers, is presented in Section V. As the name implies, an adjustment multiplier is a multiplicative factor which is used to modify the estimate obtained from a cost submodel. This approach is interesting because two projects in the *same* submodel may be multiplied by *different* adjustment factors. The estimates are more accurate because they are based on two characteristics of the project - one characteristic which determines the submodel and another which determines the adjustment multiplier. Finally, we attempted to substantiate our results using a second set of data from commercially developed software systems [Kem87]. This confirmatory data is presented in Section VI.

In the remainder of this section we provide a broad classification of cost models which leads, in Section II, to a description of the three cost models used in our study. Section III details the historical project data as well as the measures and graphs used to assess the accuracy of the cost models when calibrated to this data. Section IV - VI were described above. Section VII summarizes the results of the study.

The cost estimation method which has emerged in the computer industry is predictive cost modeling. Predictive cost models are equations which generate cost, effort or duration information from some number of project parameters. The parameters which are input to a model can include project size, complexity, staff size, environment and many more. Predictive software cost models can be divided into three classes: single independent variable (SIV) models, multiple independent variable (MIV) models and theoretical models. This classification is similar to the classification used by Basili [Bas80].

Single Independent Variable (SIV) Models

Cost models in the SIV class are based on project size as measured by thousands of source lines of code (KSLOC). All SIV models use KSLOC as the independent variable, and are empirically formed. The class can be subdivided into two groups.

Models in the first group determine an estimate of the total effort, measured in man-months, to develop a software system of a given size. These models are also called "power models" because the model equation is of the form:

$$\text{Effort} = a \cdot \text{KSLOC}^{b}$$

Regression techniques are used to determine the values of a and b which yield a best-fit to the historical data. In the power model, exponents larger than 1 model the proportionately greater difficulty of producing increasingly larger systems. For example, with an exponent of 2, doubling the system size would imply increasing the effort by a factor of 4.

Shown in Table 1 below are the equations for several different models which have been calibrated for different sets of historical data. As can be seen from this table, there are significant differences in the models' multipliers and exponents. The effect of these model differences can be seen in the rightmost column of the table which shows each model's effort estimate for a system with 100,000 lines of source code. The range of estimates extends over an order of magnitude. This wide range of estimates underscores the critical importance of calibrating a cost model to a specific application environment.

Table 1: A Comparison of Effort Estimates

Model Equation for Effort	Source	Effort Estimate for KSLOC=100
$5.2 \times \text{KSLOC}^{0.91}$	Walston, Felix IBM [Wal77]	343
$1.4 \times \text{KSLOC}^{0.93}$	Software Engineering Lab. Univ. of Maryland [Bas78]	101
$2.4 \times \text{KSLOC}^{1.05}$	Boehm (Organic Mode) TRW [Boe81]	302
$3.0 \times \text{KSLOC}^{1.12}$	Boehm (Semidetached Mode) TRW [Boe81]	521
$3.6 \times \text{KSLOC}^{1.20}$	Boehm (Embedded Mode) TRW [Boe81]	904
$0.3 \times \text{KSLOC}^{1.83}$	Schneider [Moh81]	1371

The second type of SIV models divide the system into specific types of code. The KSLOC for each type is then multiplied by a constant for that type to calculate effort. The individual effort values are then added up to give effort required for the overall project.

The Wolverton model is the best known of these models. Wolverton uses the code categories: control, input/output, pre/post-processor, algorithm, data management, and time critical. Additionally, for each code type he has an easy, medium and hard constant to represent the difficulty of that type of code. This technique can be used to create a range or to accommodate varying difficulty within a project [Put77].

In the Kustanowitz model [Moh81] the same code categories are used. However, the model uses an upper and lower productivity constant rather than complexity multipliers to get effort. After adding up the individual effort ranges a final best/worst range is created.

The Aerospace model [Moh81] uses slightly different code categories which are: real-time, nonreal-time system software and nonreal-time operational software (interactive, on-line, mathematical and string manipulation). It also includes variables for system complexity and development environment.

The Aron model is one of the earlier models and uses the programmer productivity rate philosophy. The code is divided into modules by code type. Module difficulty (easy, medium, difficult) is determined by the number of interactions the module has with other modules. The difficulty and expected duration are then used to determine the productivity rate from a table of values. The estimated size is divided by the rate to give the needed effort. This process is done for all modules giving an overall project effort estimate [Sho79].

Multiple Independent Variable (MIV) Models

Models in the MIV class use two or more independent variables in an empirically formed model. This class also has two approaches. The first approach uses ratings for such subjective factors as programmer experience, complexity, and development environment reliability. These factors are termed subjective because the "true" values for these factors cannot be determined by measurement even after the project has been completed. The other approach is an objective method which uses estimates of such objective factors as size, average staff size and language.

The subjective MIV models which appear in literature are the Price-S system and the Intermediate COCOMO model. The Price-S system was developed by RCA in the mid 70's as a byproduct of their cost estimating system for military hardware. The inputs for Price-S system are: SLOC, type of code, level of new/old code, experience and ability of staff, project difficulty (impactable factors), specifications and requirements and hardware resources. These factors are then used in a proprietary parametric equation.

The Price-S system has two other modes of operation. ECIRP mode allows the model to run in reverse based on groups' past projects. This option allows a manager to get a better understanding of the development environment. The Design-To-Cost mode allows estimators to see the impacts of trying to meet specified funding levels. Price-S has been used by the

Avionics Division of Honeywell Inc. [Fre79], together with the Putnam model by General Electric's Space Division [Jun79], and was the primary model of the Department of Air Force Space and Missile Systems Organization estimating methodology, which also includes the Doty model, the Walston and Felix model and the Putnam model [Las79]. The above three groups have reported good initial results with the Price-S system but have supplied no quantitative results.

The Intermediate COCOMO model by Boehm uses the base equations of Basic COCOMO for the initial effort estimate. The user must then determine a series of cost drivers in four areas. The four areas are product attributes, hardware attributes, personnel attributes and development environment attributes. Each driver is essentially a multiplier which can increase, decrease or not affect the estimate. Studies of the overall package of Boehm models is reported in [Boe81] [Bry83]. Other models in this group include the System Development Corporation (SDC) model completed in the early 1960's for the U.S. Air Force, the Farr and Zagorski model, and the Naval Air Development model [Moh81].

Objective MIV models are less common. The cost estimating model derived from Halstead's software science metric was the first model of this area. The inputs for the Halstead model include volume (V) and a language level value (L) [Hal77].

Thebaut's Cooperative Programming Model (COPMO) is the other objective MIV model. Thebaut's model uses two components to estimate effort: programmer cost and the cost of coordinating the programmers. Programmer cost is determined by size and coordination cost is determined by the average staff size. The details of this model and its respective equations are discussed in Section II.

Theoretical Models

Theoretical models are the class of models which are not empirically derived. These models are based on a logical framework which attempts to explain project behavior. The logical framework used in both of the models of this type is the life cycle curve of development by Rayleigh [Nor77]. Putnam's model is the most well known of the theoretical models. Putnam's model attempts to quantify the effort needed at a given point in time, from design through maintenance. Full details of the Putnam model are presented in Section II [Put77].

Parr's model is another notable model of the theoretical class [Par80]. The Parr model is also an implementation of Rayleigh's life cycle curve, with some minor differences. The primary difference is that the Parr model does not have effort start at zero due to the high level planning done before a project is initiated.

II. Selection of The Models

This section gives a detailed explanation of the three effort estimation models used in this study and describes the motivations for selecting these models. In an effort to represent the three classes of models, a model from each was chosen. The selection was also affected by the sponsoring organization's past use of the Putnam model. The three selected models were: Boehm's basic COCOMO model (a SIV model), Thebaut's COPMO model (an objective MIV model), and Putnam's model (a theoretical model). A subjective MIV model could not be selected. Models of this type require information on complexity or programmer experience which was not available in the sponsor's database. Each of the three selected models is explained in greater details below.

Boehm's Basic COCOMO Model

Boehm's Basic COCOMO model was selected to represent the single variable class. The Basic COCOMO model consists of an effort predicting equation and a duration predicting equation and can be used on three types of software. The form of the effort predicting equation is:

$$\text{Effort} = a \cdot \text{KSLOC}^{b}$$

where the values a and b are constants calculated with a best fit method.

Boehm calibrated three different sets of coefficient depending on the "mode" of the software being developed. The values which Boehm found for a variety of collected data are shown in Table 1. Organic mode software is characterized by basic adherence to requirements, minimal use of complex or innovative architectures or algorithms, and a team of experienced programmers who are familiar with this particular type of software. The size of organic mode software is less typically than fifty KSLOC. An example of an organic mode project is a simple payroll system.

Embedded mode is the other end of the spectrum and is characterized by a mixture of experience in programmers, strict adherence to requirements, as well as a complexity of external factors (hardware, regulations and procedures) which allow little flexibility in software. Embedded mode projects have a strict need to finish on schedule and these projects vary greatly in size. Embedded projects most often require innovative or complex approaches to solve the problems. An example of a embedded project is a large new operating system.

The third mode of software is referred to as semidetached mode. The characteristics of semidetached mode are a primarily experienced team on a slightly complex project with a fairly strict need to adhere to requirements. A semidetached project could also be one which is composed of some organic traits and some embedded traits. An example of a semidetached project is a simple command control system.

Thebaut's COPMO Model

The multivariable class model selected was the Cooperative Programming Model (COPMO) by Thebaut. Thebaut's approach includes two methods for predicting effort and duration. The overall emphasis of this approach is a two part model:

$$\text{Effort} = E_p + E_c$$

The first portion of his model E_p represents the productive effort (design, development, and testing) expended by the programmer. E_p is calculated as

$$E_p = a + b \cdot \text{KSLOC}$$

where a and b are constants calculated with a best fit method. The second portion of the model E_c represents the cost of coordinating multiple programmers' efforts. E_c is calculated as

$$E_c = c \cdot P^d$$

where P is the average number of personnel, and c and d are constants calculated with a best fit method.

As with other MIV models, the COPMO model incorporates the simple intuition that the effort required to develop a system increases with the system's size. The COPMO model, in addition to the size component, explicitly accounts for the communication and management overhead associated with large development staffs. This factor is only implicitly considered in other models. The exponent on P allows the model to reflect Brooks' law of programmer interaction which states that the number of possible communication paths increases quadratically with the number of programmers [Bro75].

Putnam's Macroestimating Model

The model used from the theoretical class of models is the Putnam's Life cycle model. Putnam's model is based on the life cycle curves developed by Norden [Nor77]. The basic equation which represents the life cycle curve is:

$$Y(t) = \frac{K}{D^2} * t * e^{\frac{-t^2}{2D^2}}$$

where: Y(t) is the percentage of the total manpower used,

K is the total man-months,

D is the overall duration of the project,

t is the point in time.

An equation which predicts overall effort using size and duration can be derived from the life cycle equation. The format of the effort predicting equation is:

$$\text{Effort} = c \cdot \frac{\text{KSLOC}^3}{\text{Duration}^4}$$

where c is calculated through a best fit method.

The underlying assumption of the Putnam approach is that the amount of effort required at any point in a project can be represented by the curve described by the "software equation" given above. This equation implies, for example, that a project with constant staffing throughout has wasted effort initially and uses more effort over a longer period than is needed to complete the project [Put77]. The equation also models the incompressibility of project schedules. For example, reducing the development time by 50% only reduces the total effort by approximately 6%.

III. Historical Data, Measures and Graphs

The sponsoring organization for this study was a major computer manufacturer. Their goal was to find an effort estimating model which best predicts the behavior of their development environment. An effort to collect project data began approximately fifteen years ago. The collected data was based on a variety of software projects for defense applications and contained a broad spectrum of project information. Each project is composed of one or more products. The database contains the type of product, language, estimated size, breakdown of code (new, modified and unchanged), duration, documentation pages, effort (man-months) per phase of development, and computer-time cost for each product. It is apparent from the sketchiness of some portions of the data that in these cases, the data was added after development, or that managers did not fully understand what information was needed for certain entries. Those projects which were poorly reported were not considered in our study. The resulting database of historical project data contained thirty-four projects composed of fifty-five products.

The effort information represents the product life cycle from design through integration. There are five different product types. Examples of these could be operating systems software or data compression software. The language field has three different possible entries. These entries are: HO (high order), AS (assembler) or SL (signal processing language). The database also distinguishes between new code, changed code, modified-retained code and unmodified code. New code represents code which requires design and development. Changed code represents code that was previously developed but requires modification to be used. Modified-retained code is code which was previously written for another product but required examination and little or no modification to be used in the new product. Unchanged code represents up to an entire product which was used in another product and requires no modifications to fit into the new product or project [Cru82].

Normalized Measures

To develop a single effort model it was necessary to define a normalized unit of product size which accounts for differences in the amount of reused code and the differences in languages. the sponsor of this study already had a measure called Equivalent Source Lines Of Code (ESLOC, KESLOC is one thousand ESLOC) which normalizes different code types [Cru85]. This measure is defined as:

$$
\begin{aligned}
KESLOC = & \text{ New } KSLOC + \text{ Changed } KSLOC \\
& + (0.31 * \text{ Modified Retained } KSLOC) \\
& + (0.04 * \text{ Unmodified } KSLOC).
\end{aligned}
$$

In addition, the sponsor also had a measure for normalizing two products of different languages. The measure is expressed in units of Words Of Memory (WOM, one thousand WOM is a KWOM) [Cru85]. This measure is defined as:

$$KWOM = (1.2 \; HO \; KESLOC) + (1.2 * SL \; KESLOC) + AS \; KESLOC$$

The coefficients in these equations were determined by the sponsoring organization based on their own prior empirical cost estimation studies.

Accuracy Measurements

There are many ways of measuring the accuracy of a cost model's effort estimates. Typically a number of measures are used since no single measure is sufficient to represent the full relationship between the model's estimates and the observed data. As in other similar studies [The83, Kem87] we used both numerical measures and graphical displays. Three of the numerical measures are measures of error, either absolute (*Error*)or relative (*RE* and *MRE*). A fourth numerical measure (R^2) and the graphical display represent the overall trend between the estimates and the observed data values.

Error is the simplest of the measures but a good overall indicator of the accuracy of a model. It is defined as:

$$Error = \frac{\sum_{i=1}^{n} | Actual_i - Estimate_i |}{n}$$

where n is the number of data points.

RE (average Relative Error) is a means of determining whether a prediction equation primarily overestimates or underestimates. The equation for this measure is:

$$RE = \frac{1}{n} * \sum_{i=1}^{n} \frac{(Actual_i - Estimate_i)}{Actual_i}$$

where n is the number of data points. The model is overestimating when *RE* is negative and underestimating when *RE* is positive. The ideal value for *RE* is close to zero (either positive or negative) [The83].

The *MRE* (Average Magnitude of Relative Error) is a measure of the percent error over the whole set of points. This measure is defined by the equation:

$$MRE = \frac{1}{n} * \sum_{i=1}^{n} \frac{| Actual_i - Estimate_i |}{Actual_i}$$

where *n* is the number of data points.

Each of these three measures of error captures a different aspect of the overall relationship between the estimates and the observed data and is useful to different individuals involved in cost estimation. The *Error* measure is meaningful to a project manager who needs some indication of the additional effort which may be needed on a project. For example, this measure might indicate that, on the average, the estimate may be in error by 20 person-months. The *RE* measure is useful to a corporate-level manager who is estimating the effort over many projects undertaken by the company. Even if the total *Error* is large, some projects may be significantly overestimated while others are significantly underestimated. However, if the *RE* measure is close to zero, the *total* effort estimate will be a good approximation to the total actual effort. The *MRE* measure is useful to a project manager because it is expressed as a percentage of the required effort. For example, an *MRE* of .10 means that the estimate vary from the actual by 10%. This relative measure helps to discriminate between errors made on large projects from errors made on small projects.

R^2, the Coefficient of Multiple Determination, is a numerical measure which, along with the graphical display discussed next, was used to assess the overall trend between the estimates and the actual data values. R^2 is a measure of the linearity between the actual and estimated values. The value of this measure is relative since it does not reflect how closely the values correspond. However, it is a measure of how well the prediction equation is following the data points [The83]. The formula for this equation is

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (Actual_i - f \ (Estimate_i))^2}{\sum_{i=1}^{n} (Actual_i - \mathbf{Actual})^2}$$

where *n* is the number of data points, *Actual* is the mean of the *Actual*$_i$'s, and *f* is that linear function of *Actual* which maximizes the result.

The accuracy of the model will also be depicted in graphs which display the estimated effort versus the actual effort. A perfect estimator would be one in which all of the data points lie on the line Y=X. For reference each graph will also show the lines Y=X, Y=.75*X, and Y=1.25*X. The last two lines represent a region of 20% error around the perfect estimator. This region is shown because, as in [The83], our criteria for an acceptable effort estimator is that 75% of the estimates lie within 25% of the actual values.

IV. Model Comparison and Improvements

Our first goal was to compare the accuracy of each of the three models. A best-fit correlation was done for each model over all products and for each model over all projects. This generated the constants for the estimating equations in Table 2. Table 3 gives empirical results on the accuracy of the estimates obtained from the equations in Table 2.

Table 2. Single Product and Project Model Equations

COCOMO

$$\textit{Product}: \quad \textit{Effort} = 3.826 * \textit{KWOM}^{1.0}$$
$$\textit{Project}: \quad \textit{Effort} = 4.088 * \textit{KWOM}^{1.0}$$

COPMO

$$\textit{Product}: \textit{Effort} = 47.953 + 0.279 * \textit{KWOM} + 2.017 * \boldsymbol{P}^{1.675}$$
$$\textit{Project}: \textit{Effort} = 113.133 + 0.771 * \textit{KWOM} + 0.369 * \boldsymbol{P}^{2.00}$$

Putman

$$\textit{Product}: \textit{Effort} = 1.677 * (\textit{KWOM}^{3} \ / \ \textit{Duration}^{4})$$
$$\textit{Project}: \textit{Effort} = 1.760 * (\textit{KWOM}^{3} \ / \ \textit{Duration}^{4})$$

Table 3. Results for Single Product and Single Project Models

		Error	*RE*	R^2	*MRE*
COCOMO	Product :	196.5	-0.43	0.66	0.85
	Project :	308.9	-0.56	0.66	0.97
COPMO	Product :	116.0	-0.57	0.93	0.86
	Project :	165.7	-0.93	0.96	1.19
Putnam	Product :	295.8	0.91	0.11	0.91
	Project :	477.4	0.90	0.09	0.95

There are two main observations to be made about the results in Table 3. First, the Putnam model is clearly the least accurate of the three models. The Putnam R^2 measures are particularly low and there is no compensating strength evident in its *RE* and *MRE* measures.

While the COPMO model has very strong R^2 values its *RE* and *MRE* measures are slightly worse than for the COCOMO model. The major drawback of the COCOMO model is the *Error* in comparison with COPMO. Additional data will have to be examined in order to make a judgement about the relative merits of the COCOMO and COPMO models.

The second observation is that none of the three models, either at the product or project level, seems to provide acceptable estimates. For example, the MRE values range from 0.85 to 1.19. The lack of accuracy can also be seen in Figure 1 (dotted line) which shows clearly that many of the product estimates from the COPMO model lie outside our target range of (-25%,+25%). We, therefore, explored two methods of improving the accuracy of the models. These two techniques, termed submodeling and constructive estimation, are presented below.

<<insert Figure 1: Graph of COPMO Model Estimates vs. Actual Effort>>

Submodeling

One possible reason for the model's poor estimates lies in the lack of any highly regular pattern in the database between size and effort. For the fifty-five products *KWOM* ranged from 2.7 to 225 and effort ranged from 18 to 1300 man-months. For thirty-four projects *KWOM* ranged from 2.7 to 1067, and effort ranged from 18 to 7106 man-months. The lack of a simple relationship between size and effort implies that the projects/products may need to be divided into subgroups so that there is a more consistent relationship within each each subgroup. Dividing the data into groups and generating a model for each group is called submodeling.

Four attempts to divide the data in logical groupings to achieve more accurate results were performed. The groupings were by *Language,* by *Product type,* by *Size,* and by *Duration.* The best results were produced by the Duration submodel and are presented next. The interested reader is invited to read [Gin86] for the other submodel results.

To form a duration submodel the products are divided into groups based on the duration of the product development. The five classes in the duration submodel and the number of data points for each class are shown in Table 4. Once again the constants for each submodel were generated with a best-fit method for each of the three models for all five classes. The results are shown in Table 5.

Table 4. Duration Submodel Equations

Class 0	(Elapse < 10) (8 products)		
		COCOMO:	$Effort = 3.670 * KWOM^{1.0}$
		COPMO:	$Effort = 3.783 + 1.537 * KWOM + 3.710 * P^{1.0}$
		Putnam:	$Effort = 2.999 * KWOM^{3} / Duration^{4}$
Class 1	(10 ≤ Elapse < 20) (10 products)		
		COCOMO:	$Effort = 4.624 * KWOM^{1.0}$
		COPMO:	$Effort = 25.130 + 1.346 * KWOM + 1.028 * P^{1.8}$
		Putnam:	$Effort = 58.108 * KWOM^{3} / Duration^{4}$
Class 2	(20 ≤ Elapse < 30) (19 products)		
		COCOMO:	$Effort = 3.356 * KWOM^{1.0}$
		COPMO:	$Effort = 12.076 - 0.301 * KWOM + 25.441 * P^{1.0}$
		Putnam:	$Effort = 2.999 * KWOM^{3} / Duration^{4}$
Class 3	(30 ≤ Elapse < 40) (11 products)		
		COCOMO:	$Effort = 1.729 * KWOM^{1.0}$
		COPMO:	$\text{Effort} = 7.469 - 0.193 * KWOM + 34.386 * P^{1.0}$
		Putnam:	$Effort = 1.408 * KWOM^{3} / Duration^{4}$
Class 4	(Elapse ≤ 40) (7 products)		
		COCOMO:	$Effort = 3.716 * KWOM^{1.0}$
		COPMO:	$\text{Effort} = -56.491 + 1.441 * KWOM + 36.945 * P^{1.04}$
		Putnam:	$Effort = 1.408 * KWOM^{3} / Duration^{4}$

Table 5. Duration Submodel Results

Class	Model	*Error*	*RE*	R^2	*MRE*
0	COCOMO	52.9	.27	.66	0.47
	COPMO	27.6	-.26	.78	0.42
	Putnam	68.9	.68	.44	0.80
1	COCOMO	29.6	-.07	.89	0.22
	COPMO	13.6	-.03	.96	0.11
	Putnam	84.7	.68	.43	0.77
2	COCOMO	106.3	-.20	.83	0.62
	COPMO	30.0	-.08	.96	0.15
	Putnam	210.1	.90	.25	0.92
3	COCOMO	132.8	-.21	.81	0.98
	COPMO	11.9	-.02	.99	0.19
	Putnam	151.6	.90	.79	0.90
4	COCOMO	130.6	-.72	.99	1.01
	COPMO	11.7	-.04	.99	0.10
	Putnam	269.1	.83	.97	0.83
Combined	COCOMO	92.8	.17	.97	0.64
	COPMO	20.7	-.08	.99	0.18
	Putnam	119.2	.82	.92	0.86

Table 5 shows that the COPMO model is unquestionably the best predictor of the three models considered. Overall, the COPMO model predicted with an *ERROR* measure of just over 20 MM, or an *MRE* of about 18%. This accuracy is competitive with the best claimed in the literature for models of this kind. The only drawback of this model is the need for prediction of not only size but also duration. However, given the accuracy of the model and the fairly broad sizes of the classes, it is a model worth further investigation.

The accuracy of the COPMO duration submodel is confirmed by the graph shown in Figure 1 (un-dotted line). The values in this graph lie close to the X=Y line and have only a small number of values outside the 25% area. Furthermore, almost 90% of the *MRE* values within 40% error and 95% of the *MRE* values within 60% error for the COPMO duration submodel.

Constructive Modeling Technique

It is also apparent that the submodeling approach is more successful than the single model approach for all three models. This can be seen graphically or the COPMO model in Figure 1 and numerically for all models by comparing the combined measures in Table 5 with the corresponding product measures given in Table 3. We see, for example, that the single COCOMO model has *ERROR* = 196.0 and *MRE* = .85 while the duration submodel reduces these measures to 92.8 and .64, respectively. Improvement for the Putnam model can also be seen.

A second technique to improve the accuracy of the models attempts to exploit the project-product structure. Recall that a project consists of multiple products. A comparison of the results of the single product model versus the single project model (Table 3) reveals that estimation done at product level is more accurate than at the project level. However, much contracted software is generally of a size requiring subdivision of the project into multiple products. The product estimates can be underestimates as easily as overestimates. If product effort estimates were summed to form a project estimate, there can be a cancelling effect to reduce the error in the final estimate. For example, given a project Z which is composed of two products X and Y, estimates are generated for both products. Suppose that the effort estimate for product X is 10 MM low, and the effort estimate for product Y is 10 MM high. If the effort estimate for Z is obtained by summing the estimates for X and Y, the error would be zero. Summing up product estimates to form a project estimate is called the Constructive Modeling Technique (CMT).

To test the CMT, we selected from the database all multi-product projects. There were six projects consisting of a total of twenty-seven products. The number of multi-product projects, although small, seemed sufficient to attempt the technique. The test of CMT was conducted in the following manner. First, we recorded the six project estimates from the overall model. We then obtained estimates for each of the twenty-seven projects. All projects within the same product were then summed to obtain the CMT project estimate. To further explore the submodeling strategy we obtained the product estimates from each of the four submodels. To measure the improvement of CMT we used summed absolute error (SAE) which is defined as:

$$SAE = \sum_{i=1}^{n} | Actual_i - Estimate_i |$$

The results of the comparison are shown in Table 6.

Table 6. Results of CMT on Multi-product Projects

Submodel	*SAE*	*SAEw/CMT*
Language	1643	1134
Product Type	1088	736
Size	972	533
Duration	327	162

In each case of the submodel estimates being added together to form a set of project estimates, the *SAE* was noticeably decreased. Although there is only a small number of projects to test the method, the consistent improvement by the CMT shows its potential value. These results also reflect again that, at least for the database we used, the Duration Submodel provided the most accurate estimates of the four submodeling strategies.

V. Adjustment Multipliers

The duration submodel presented above generates poor estimates for some products. This can be seen for the COPMO duration submodel in Figure 1 where there are several data points outside of our target region. Adjustment multipliers is a method which attempts to correct the poor estimates, without affecting the good estimates. The data which we present in this section is limited to the COPMO model since it was the best of the three model we were considering. Similar results to those given below for the other models may be found in [Gin86].

Adjustment multipliers is a refinement of a submodel estimate. The initial effort estimate (from the submodel) is adjusted by the selected multiplier (based on the adjustment multiplier characteristic) to generate the final estimate. For example, suppose that the example product has the following characteristics: expected $KWOM = 28$, expected $P = 14$, and the expected duration is 16 months. Also suppose that we have determined that the COPMO model, a duration submodel and a size adjustment multiplier are to be used. Since the expected duration is between 10 and 20 months, the class 1 COPMO submodel is the equation used to generate the initial estimate. The following equation is from this size submodel (Table 4):

$$\text{COPMO: } \mathit{Effort} = 25.130 + 1.346^{*}\,\mathit{KWOM} + 1.028^{*}\,P^{\,1.8}$$

Using $KWOM = 28$ and $P = 14$, the equation produces an initial effort estimate of 182 MM. Next, the size adjustment multiplier is selected (from Table 7). The adjustment multiplier class is 1 (*KWOM* is between 10 *KWOM* and 30 *KWOM*), so the appropriate multiplier is 1.032. Therefore the final effort estimate is 182 * 1.032 which is equal to 188 MM.

The adjustment multipliers are generated by evaluating a best-fit correlation between the estimated values of the submodels and the actual effort values. The complete set of multipliers is available in [Gin86]. For brievity, this paper only presents one set of adjustment multipliers. The size adjustment multipliers on the duration submodel estimates are presented below. The equations and results for this set of size adjustment multipliers are supplied in Tables 7 and 8. and Figure 2.

Table 7. Size Adjustment Multipliers for Duration Submodel Estimates

Class 0: ($KWOM < 10$)
Effort = 0.689 * Duration Effort Estimate

Class 1: ($10 \leq KWOM < 30$)
Effort = 1.032 * Duration Effort Estimate

Class 2: ($30 \leq KWOM < 60$)
Effort = 1.003 * Duration Effort Estimate

Class 3: ($60 \leq KWOM < 100$)
Effort = 0.962 * Duration Effort Estimate

Class 4: ($KWOM \leq 100$)
Effort = 1.001 * Duration Effort Estimate

Table 8. Results of Size Adjustment Multipliers on the Duration Submodel

	Error	*RE*	R^2	*MRE*
Duration Submodel	20.9	-0.08	0.99	0.18
Size Adjustment Multipliers	20.0	-0.003	0.99	0.13

<<insert full page Figure 2: COPMO Adjustment Multipliers>>

The significance of the size adjustment multipliers used on the duration submodel estimates are that they represent the highest accuracy achieved in this study. Furthermore, the MRE value of 13% lies in the range of the best reported models (15%-20% [Dem82]). The improvement shown in Table 8 is worth noting since the improvement is done on a highly accurate submodel.

VI. Confirmatory Data

In the previous sections we have seen that the cost estimates derived from the COPMO model were more accurate than those obtained from either the basic COCOMO model or from Putnam's model. A major question which remains is whether this conclusion would be true for projects undertaken in an environment different from the one from which our data was obtained. To answer this question we repeated a portion of our experiments using data, also from an anonymous industrial source, which appeared in a recent paper by Kemerer [Kem87].

The first confirmatory experiment, shown below in Table 9, is a comparison of the three models. As with our own data, we observe that the COPMO model is more accurate than Putnam's model and, with the single exception of the *RE* measure, is also to be preferred over the COCOMO model. A closer look at the COCOMO measures suggests that this model has substantial estimating errors (low R^2 measure combined with large *MRE* measure) where the overestimates and the underestimates tend to cancel each other (low *RE* measure). By contrast, the COPMO model evidences less general estimating error (high R^2 measure combined with small *MRE* measure). However, in the COPMO model the overestimates and the underestimates do not cancel each other as much as in the COCOMO model leading to a higher *RE* measure. The lower total error of the COPMO model also confirms it better overall estimating capability.

Table 9. Model Comparison using Kemerer's Data

	Error	*RE*	R^2	*MRE*
COCOMO	115.7	.07	.75	.67
COPMO	46.0	-.23	.95	.43
Putnam	161.6	.45	.48	.69

Two difference between our measures and those used by Kemerer should be pointed out. First, different regression measures are used. Kemerer uses a "more conservative" measures that is "... adjusted for degrees of freedom, which results in slightly (sic) lower values than R^2." This explains why the regression measure reported by Kemerer for the COCOMO model is 0.68 while the value shown in Table 9 above is 0.75. Second, each model which we use has been calibrated to the specific data used in each case. For example, the COCOMO model was calibrated to our own data (resulting in the set of equations shown in Table 2) and calibrated independently (resulting in a different set of equation constants) to Kemerer's data. Kemerer's results, on the other hand, are based on a combination of calibrated and uncalibrated models.

The second confirmatory experiment tested the submodeling technique presented in Section IV. The results of applying this technique to Kemerer's data are presented in Table 10. Kemerer's data contains 15 projects. Because of the small number of projects only two duration submodels were formed based on a division of the data into two groups containing 7 and 8 projects, respectively. This division placed into one class those projects with duration not more than 13 months and into a second class those projects with duration longer than 13 months. The top two major divisions of Table 10 show the accuracy of each submodel on its own set of data points. The last major division of the table ("Combined") shows the overall accuracy of the estimation technique when the estimates from the two submodels are composed and evaluated as a whole.

Table 10. Confirmation of Duration Submodel Results

Class	Model	*Error*	*RE*	R^2	*MRE*
Duration <= 13 (7 projects)	COCOMO	27.0	-.15	.96	.41
	COPMO	15.8	-.02	.96	.12
	Putnam	73.1	.55	.74	.60
Duration >13 (8 projects)	COCOMO	110.5	.10	.91	.71
	COPMO	32.3	-.12	.98	.25
	Putnam	149.6	.34	.84	.87
Combined	COCOMO	71.5	-.02	.92	.56
	COPMO	24.5	-.07	.98	.19
	Putnam	113.8	.43	.83	.74

Table 10 confirms two main conclusions of our study. First, it shows again that the COPMO model provides better estimates than the other two models. Second, a comparison of the combined duration submodel results with the measures in Table 9 (no submodeling) demonstrates the value of the submodeling technique. For example, the submodeling reduces the COPMO *Error* measure from 46.0 to 24.5 and also reduces the *MRE* measure from .43 to .19.

We were not able to perform a confirmation experiment of the adjustment multipliers because Kemerer's data was much smaller than our own data (15 projects vs. 34). Unfortunately, this smaller size prevented the application of the adjustment multiplier technique because each group would have had too few data points to yield a meaningful result. Nor does Kemerer's data provide any project/product breakdown necessary to apply the constructive modeling technique.

VII. Conclusions

Within the realm of software estimating models, this study has accomplished a variety of valuable tasks. The primary accomplishment is the comparison of Thebaut's COPMO model against Boehm's organic COCOMO model and Putnam's model (Section IV). The COPMO results reveal that it consistently performs as well as, and usually better than the COCOMO and Putnam models (Tables 3, 5, 9 and 10). The Basic COCOMO model by Boehm shows varied performance throughout the study. Boehm's model only occasionally matches the performance of the COPMO model. The Putnam model, centered around the equation which predicts the needed effort at a given point in time during the project life, produced estimates which were worse, sometimes far worse, than those of the other two models.

The second major result of our study is the use of two techniques to improve the estimates obtained from the basic models (Section IV). Application of submodeling (Table 5, Figure 1) and constructive modeling (Table 6) improved the estimates of all three models. The best product level models are the COPMO duration submodels. The best project level model is the constructive modeling technique used on the COPMO duration submodels. Both the best project model and best product model have the *MRE* within twenty-five percent which is considered acceptable according to the current modeling literature [Dem82]. The COCOMO model shows its best results for three out of the four submodels: language, product and duration, only one of which was presented in this paper. For all three submodels the *MRE* is in the 60 to 70 percent range which is not acceptable but is still considerably better than the Putnam model. The other characteristic behavior of the COCOMO model is that its *RE* value is usually close to zero. This characteristic implies that the model tends to equally overestimate and underestimate. Although the characteristic may seem insignificant, it could be highly advantageous when used to form constructive project estimates since often the overestimates would cancel underestimates.

The third major result is the definition and exploration of a new technique, termed adjustment multipliers (Section V). The use of this technique resulted in further improvements in the COPMO model's estimates (Table 8, Figure 2). This technique yields the best results in our study because it allows an estimator to "tune" an initial estimate by taking into account two factors. In our study duration and size were the most sensitive factors. Although we have used only three cost models, it should be noted that adjustment multipliers is a general technique which can be applied to any model.

Finally, we used a second set of industry data (Section VI) to confirm that the COPMO model provides better estimates than the COCOMO or Putnam models (Table 9) and that the submodeling technique yielded improved estimates (Table 10). Limitations on this second set of data prevented the confirmation of the constructive modeling technique and the adjustment multipliers. However, the fact that the submodeling technique was successful on this second set of data suggests that the adjustment multipliers would also be useful in cases other than the one in which we have used it.

References

[Aro77] Aron, J. D., "Estimating Resources For Large Programming Systems," COMPSAC77 Tutorial, IEEE, November 1977, pp. 257-268.

[Bas78] Basili, V. R. and Zelkowitz, M. V., "Analyzing Medium-Scale Software Development," Proceedings, 3rd International Conference On Software Engineering,1978, pp. 116-123.

[Bas80] Basili, V. R., "Resource Models," Tutorial on Models and Metrics For Software Management and Engineering, IEEE, 1980, pp. 4-9.

[Bau79] Bauer, T. H., "Software Cost Estimating Experience," Proceedings: IEEE Workshop on Quantitative Software Models, IEEE, October 1979, pp. 78-81.

[Boe78] Boehm, B. W. and Wolverton R. W., "Software Cost Modeling: Some Lessons Learned," Proceedings: Second Software Life Cycle Management Workshop, U.S. Army CSC and IEEE, August 1978, pp. 129-132.

[Boe81] Boehm, B. W., Software Engineering Economics, Prentice Hall Inc., Englewoods Cliffs, NJ., 1981.

[Boy84] Boydston, R. E., "Programming Cost Estimate: Is It Reasonable?", Proceedings: 7th Int. Conference On Software Engineering, ACM, IEEE and National Bureau of Standards, 1984, pp. 153-159.

[Bro75] Brooks, F. P., The Mythical Man-Month, Addison-Wesley Inc., Reading, MA, 1975

[Bru82] Bruce, P. and Pederson, S. W., The Software Development Project -- Planning and Management, John Wiley and Sons Publishing Inc., New York, NY., 1982.

[Bry83] Bryant, A. and Kirkham, J. A., "B. W. Boehm Software Engineering Economics: A Review Essay," ACM SIGSOFT Notes, Vol. 8, No. 3, July 1983, pp. 44-59.

[Buc77] Buckle, J. K., Managing Software Projects, American Elsevier Publishing Company Inc., New York, NY., 1977.

[Cru82] Cruickshank, R. D., and Lesser, M., "An Approach to Estimating and Controlling Software Development Costs," in R. Goldberg (ed.), The Economics of Information Processing, Volume 2, John Wiley & Sons, Inc., 1982, pp. 139-148.

[Cru85] Cruickshank, R. D., personal correspondence, September 1985.

[Dem82] DeMarco T., Controlling Software Projects, Yourdon Press, New York, NY., 1982.

[Eva83] Evans, M. W., Piazza P. and Dolkas J. B., Principles of Productive Software Management, John Wiley and Sons Inc. New York, NY., 1983.York, NY., 1983.

[Fox82] Fox, J. M., Software and Its Development, Prentice-Hall Inc., Englewood Cliffs NJ., 1982.

[Fre79] Freiman, F. R. and Park, R. E., "The Price Software Cost Model," NAECON Proceedings, IEEE, 1979, pp. 280-288.

[Gaf79] Gaffney, J. E. Jr and Heller, G. L., "Macro Variable Software Models For Application To Improved Software Development Management," Proceedings: IEEE Workshop On Quantitative Models, IEEE, October 1979, pp. 63-68.

[Gin86] Gintner, M.G. Development of a Resource Scheduling Model, M.S. Thesis, Department of Computer Science, Virginia Tech, 1986.

[Hal77] Halstead, M. H., Elements of Software Science, Elsevier North-Holland, Inc., New York, NY, 1977.

[Han82] Hannan, J. (editor) and Oliver, P., A Practical Guide To Computer Programming Management, Auerbach Publishers Inc., Pennsauken, NJ., 1982.

[Hor75] Horowitz, E. (editor) and Wolverton, R. W., Practical Strategies For Developing Large Software Systems, Addison-Wesley Publishing Company, 1975.

[Jun79] Junk, W. S., "A Software Cost Estimation Methodology Creating The Structure For Reliable Application Of State Of The Art Cost Models," Proceedings: IEEE Workshop On Quantitative Models, IEEE, October 1979, pp. 56-62.

[Kem87] Kemerer, C. F., "An Empirical Validation of Software Cost Estimation Models," Communications of the ACM, Vol. 30, No. 5, May 1987, pp. 416-429.

[Las79] Lasher, W., "Software Cost Evaluation and Estimation: A Government Source Selection Case Study," Proceedings: IEEE Workshop on Quantitative Models, IEEE, October 1979, pp. 42-55.

[Moh81] Mohanty, S.N., "Software Cost Estimation: Present and Future," Software Practice and Experience, Vol. 11, 1981, pp. 103-121.

[Nor77] Norden P. V., "Useful Tools for Project Management," COMPSAC77 Tutorial, IEEE, November 1977, pp. 113-143.

[Par80] Parr, F. N., "An Alternative to the Rayleigh Curve Model for Software Development Effort," IEEE Transactions on Software Engineering, IEEE, May 1980, pp. 291-296.

[Put77] Putnam, L. H., "The Software Life Cycle: Practical Application to Estimating Cost, Schedule and Providing Life Cycle Control," Compsac77 Tutorial, IEEE November 1977, pp. 39-112.

[Rub83] Rubin, H. A., "Macro-Estimation of Software Development Parameters: The ESTIMACS System," SoftFair Proceedings, IEEE, National Bureau of Standards, and ACM, July 1983, pp. 109-118.

[San79] Sandler, G. H. and Rachowitz, B. I., "Software Cost Models --Grumman Experience," IEEE Workshop on Quantitative Models, October 1979, pp. 69-77.

[Sho79] Shooman, M. L., "Tutorial On Software Cost Models", IEEE Workshop On Quantitative Software Models, IEEE, 1979, pp. 1-19.

[Sho83] Shooman, M. L., Software Engineering, McGraw-Hill Book Company, New York, NY, 1983.

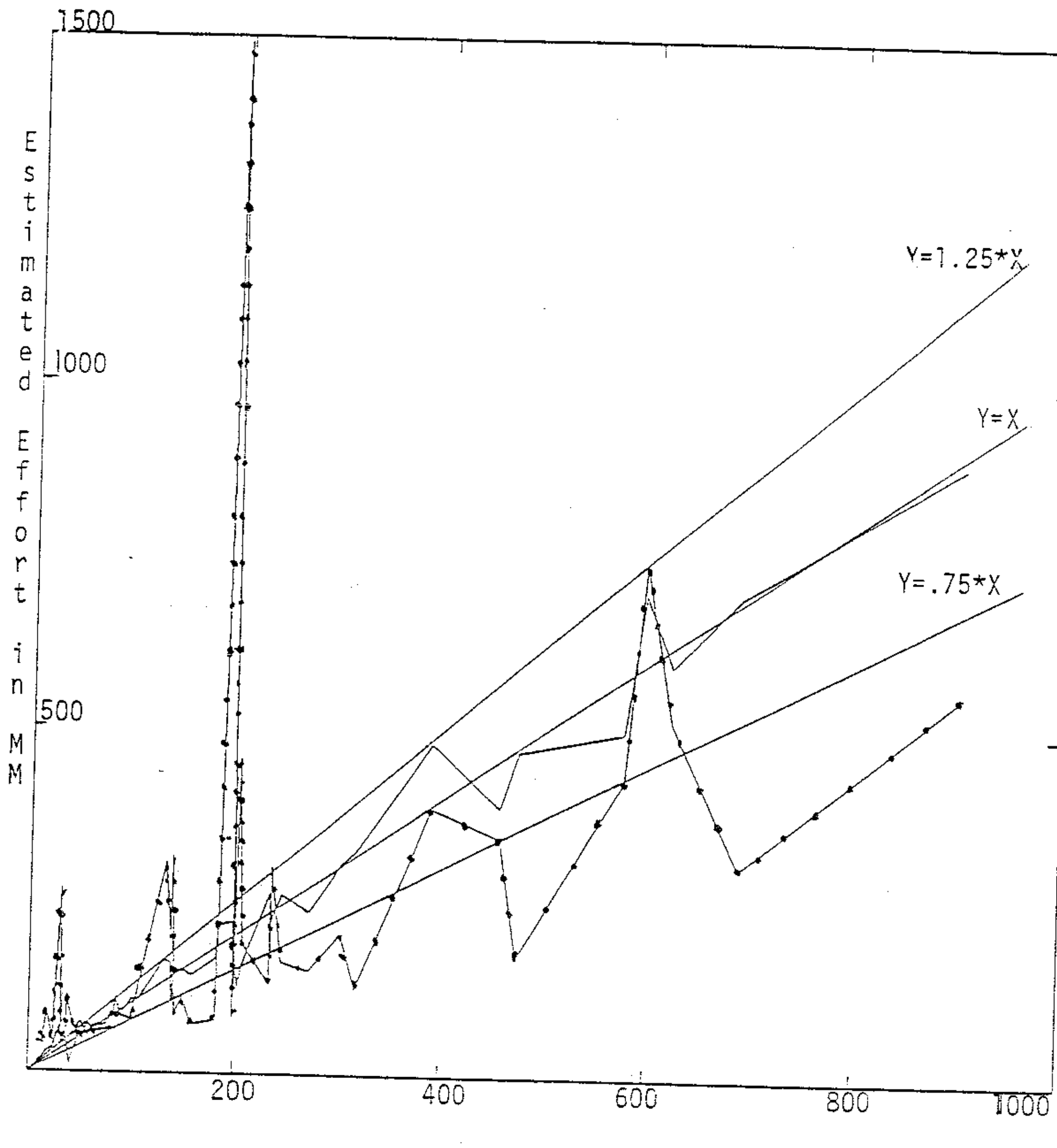
[Ste77] Stephenson W. E., "An Analysis of the Resources Used in the Safeguard System Software Development," COMPSAC77 Tutorial, IEEE, November 1977, pp. 303-312.

[The83] Thebaut, S. M., The Saturation Effect In Large-Scale Software Development: Its Impact and Control, Ph.D. Thesis, Department of Computer Science, Purdue University, 1983.

[Tur84] Turner, R., Software Engineering Methodology, Reston Publishing Company Inc., Reston, VA., 1984.

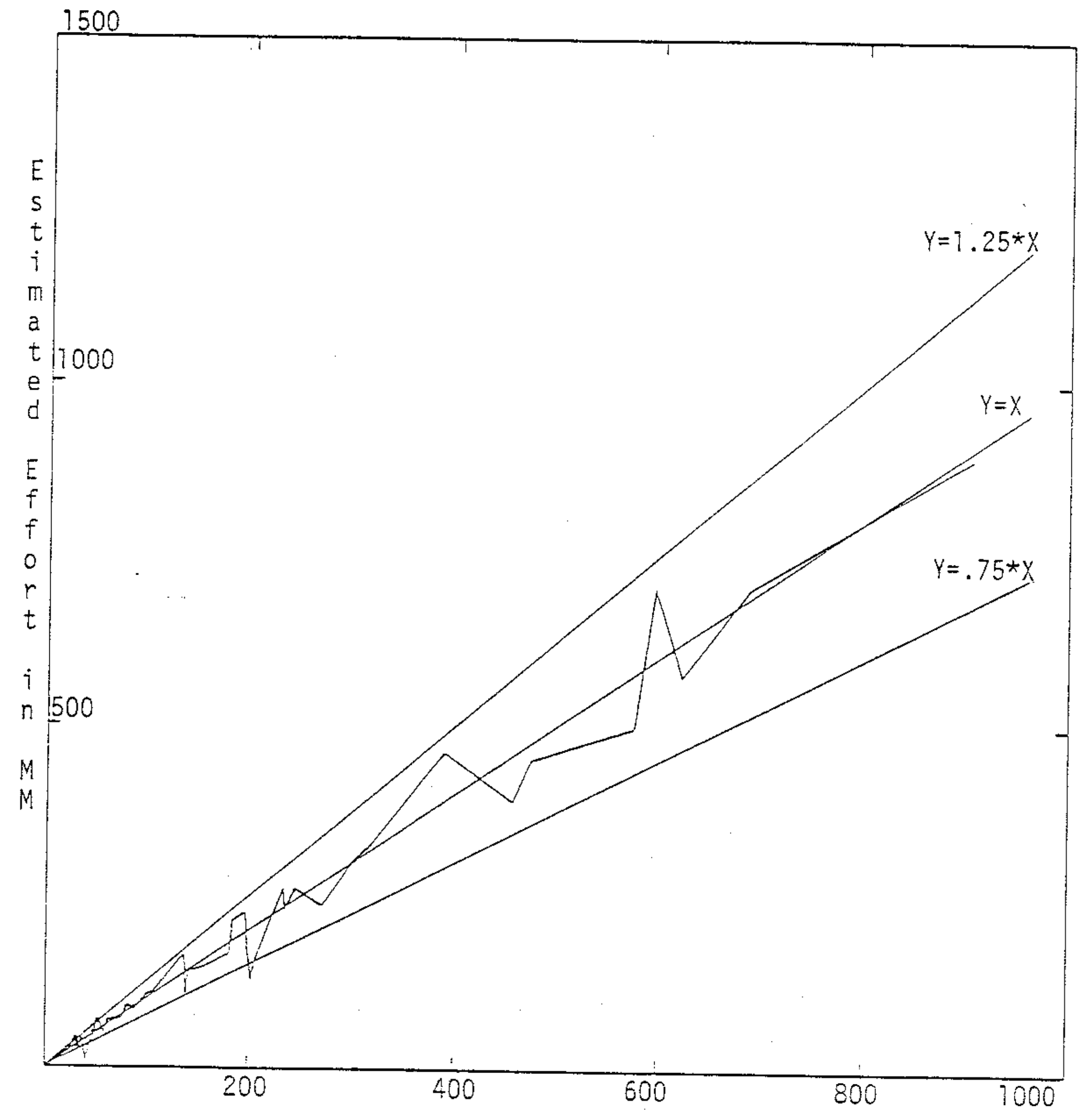
[Wal77] Walston, C. E. and Felix C. P., "A Method of Programming Measurement and Estimation," IBM Systems Journal, Vol. 16, No. 1, 1977, pp. 54-73.

[Wol77] Wolverton, R. W., "Quantitative Management: Software Cost Estimating," COMPSAC77 Tutorial, IEEE, November 1977, pp. 145-255.



1500
1000
500
Estimated Effort in MM
Y=1.25*X
Y=X
Y=.75*X
200
400
600
800
1000
Actual Effort in MM
Actual Effort vs. COPMO Duration Submodel
Actual Effort vs. COPMO Single Product Model

Figure 1



Size Adjustment Multipliers on Duration Submodel Estimates

Figure 2