# Software Cost Estimation Using Economic Production Models

QING HU, ROBERT T. PLANT, AND DAVID B. HERTZ

QING HU is Assistant Professor of Information Systems in the Department of Decision and Information Systems at the Florida Atlantic University. He received his Ph.D. in computer information systems from the University of Miami, Florida. Dr. Hu currently teaches information systems, database management, and data communications courses. His research interests include software engineering, economics of information technology, IT outsourcing, and electronic commerce. He has published articles in such journals as *Computers and Industrial Engineering*, *Information Sciences*, *California Management Review, IEEE Transactions on Software Engineering*, and *Information Systems Research*, and presented numerous papers at national and international information systems conferences.

ROBERT T. PLANT is an Associate Professor in the Department of Computer Information Systems at the University of Miami, Coral Gables, Florida. Dr. Plant received his Ph.D. in computer science at the University of Liverpool, England. Having previously studied at the Programming Research Group, Oxford University, and Wadham College, Oxford, Dr. Plant was Chairman of the Workshop on Validation and Verification of Knowledge-based Systems at AAAI 1994 and Co-Chair of the 1997 Workshop. Dr. Plant is a Chartered Engineer (U.K.), a European Engineer, a Fellow of the British Computer Society. He holds a Visiting Professorship in Computer Science at the University of Wolverhampton in England and is a Visiting Associate at Templeton College, Oxford. His research interests are in virtual organization, knowledge-based systems and software engineering. He has published in journals such as *Communications of ACM*, *Information and Management*, and *Journal of Systems and Software*.

DAVID B. HERTZ is Professor Emeritus in the Department of Computer Information Systems at the University of Miami. He received his Ph.D. in management science from Columbia University. Dr. Hertz currently is the CEO and Chairman of Identification Technologies International, Inc. He was formerly management consultant and partner of McKinsey and Co. in New York City. He has published numerous articles in the areas of operations research, risk management, and artificial intelligence in several journals, including *Management Science, Harvard Business Review, and Information Sciences*, and is the author and coauthor of many books.

ABSTRACT: One of the major difficulties in controlling software development project cost overruns and schedule delays has been developing practical and accurate software cost models. Software development could be modeled as an economic production process and we therefore propose a theoretical approach to software cost modeling. Specifically, we present the Minimum Software Cost Model (MSCM), derived from economic production theory and systems optimization. The MSCM model is compared with other widely used software cost models, such as COCOMO and SLIM, on

*Journal of Management Information Systems* / Summer 1998, Vol. 15, No. 1, pp. 143–163.

0742–1222 / 1998 $9.50 + 0.00.

the basis of goodness of fit and quality of estimation using software project data sets available in the literature. Judged by both criteria, the MSCM model is comparable to, if not better than, the SLIM, and significantly better than the rest of the models. In addition, the MSCM model provides some insights about the behavior of software development processes and environment, which could be used to formulate guidelines for better software project management polices and practices.

KEY WORDS AND PHRASES: economic production theory, software cost estimation, software cost models, software production, software project management.

---

IT WAS ESTIMATED THAT THE SOFTWARE COST to U.S. companies and government agencies would reach $225 billion a year by 1995, compared with $70 billion in 1985 [4]. The growing software cost poses a significant challenge to the software industry not only to adapt to the ever-evolving software technologies, but also to develop software application systems more efficiently with sound software project management practices, which has been a major impetus behind the move toward software engineering since the late 1960s. The effort so far has produced mixed results. On one hand, integrated CASE tools, fourth-generation languages (4GLs), and object-oriented programming technology have forever changed the way in which software systems are constructed at the micro level. On the other hand, managing software development projects with engineering precision at the macro level is still far from reality. Some significant problems that plagued the software projects in the 1960s and 1970s still exist and may even have intensified due to the increasing scale and complexity of new computer applications.

Among the worst of these problems are software cost overruns and schedule slippages. A 1984 study [13] of seventy-two software projects in twenty-three major U.S. corporations revealed that the median cost overrun is about 34 percent with an average of 67 percent, and the average schedule slippage is about 22 percent. In a survey of forty-five business software systems completed in 1987, Putnam and Myers [23] found that the average cost overrun was about $225,000 and schedule slippage was about three calendar months, and that these problems happened in all countries and that no company was immune. A study by Peat Marwick Mitchell and Co. found that more that 35 percent of the company's 600 largest customers had major software projects cost overruns and schedule slippages [24]. These statistics may just show the tip of the iceberg. Our experience with software project managers of various industries indicates that a 200 to 300 percent cost overrun and a 100 percent schedule slippage would not be unusual in large software systems development projects. Millions of dollars have been wasted in projects that were abandoned because of severe cost overruns and schedule slippages [15, 24].

While many factors could have contributed to the problems of software project management [11, 21, 28], inaccurate estimation of development cost and schedule, which leads to unrealistic expectation and project planning, is often considered one of the top contributors. As a result, many studies on software project management

have focused on the issue of developing software cost models. Despite the progress made in developing better models, accurate estimations of software project cost and schedule remain elusive. More than a decade ago, Mohanty [19] evaluated thirteen software cost models and concluded that none of those models could estimate software cost with a satisfactory degree of certainty. Later, Kemerer [16] compared several major software models (COCOMO, SLIM, ESTIMACS, and Function Points) and revealed that the average magnitude of relative errors (MRE) of the estimates using these models ranged from 85 to 772 percent, with many in the 500–600 percent range. A recent study by Jorgensen [14] compared multiple regression, neural networks, and pattern recognition approaches for estimating software maintenance effort and again found no comfort: With MREs ranging from 60 to 280 percent across all models, the best and the worst result were all produced by the multiple regression models.

What went wrong with these models? There could be many answers to this question. We conjecture that the empirical foundations for these models may have contributed to and resulted in unpredictable performance when used for software cost estimation. Thus, we postulate that a software cost model with a sound theoretical basis would provide a more stable platform from which to derive a clearer understanding of cost modeling in the future. Furthermore, a theoretically sound model could yield some insights into the fundamental behavior of software development processes that were not apparent from empirical models. In this study, we present a software cost estimation model, called the MSCM method (for Minimum Software Cost Model), based on the classic economic production theories.

## Review of Software Cost Models

OVER THE YEARS DOZENS OF SOFTWARE COST MODELS have been developed for various purposes. Some are proprietary, others are in the public domain. More comprehensive review of these models can be found in [3, 16, 19]. This section focuses on the two widely cited algorithmic models, COCOMO by Boehm [3] and SLIM by Putnam [22,23], since they are used in many studies as the benchmark models.

As one of the earlier algorithmic software cost models, COCOMO is the most widely accepted software cost and schedule estimation method. Over the years it has served as a benchmark for evaluating the performances of various cost estimation models and methods [16, 17, 20, 26].

The COCOMO model consists of three submodels: Basic, Intermediate, and Detailed COCOMO. Boehm's evaluation of the three models has concluded that the Intermediate COCOMO is significantly better than the Basic COCOMO, while the Detailed COCOMO is not noticeably better than the Intermediate COCOMO. Thus, only the Intermediate COCOMO is discussed here. It can be written as:

$$E = KS^{\alpha}\prod_{i=1}^{15} c_i, \tag{1}$$

where $K$ and $\alpha$ are parameters dependent on the mode of the software system to be

developed, $S$ is the software size measured in KDSI (thousand delivered source instructions), and the $c_i$'s are the fifteen so-called cost drivers, which are scalars ranging from 0.70 to 1.66 with a nominal value of one, reflecting the characteristics of software systems as well as the production environment, such as the required software reliability and programmer skills.

The result of the Intermediate COCOMO can be significantly influenced by the values of $c_i$'s, which require detailed information about the software system and the development environment. Since the cost drivers and the values were originally based on the software project data of the 1960s and 1970s, the validity and accuracy of the Intermediate COCOMO are at best uncertain in today's complex software environment.

The SLIM model was first proposed by Putnam [22] and then revised significantly later in his 1992 book [23]. The basic estimation equation in SLIM method is:

$$S = PP\,(E/B)^{\frac{1}{3}}\,t_d^{\frac{4}{3}}, \tag{2}$$

where $S$ is the software size in SLOC (source lines of code), $PP$ is a productivity parameter, $E$ is the effort measured in person-years, $B$ is a skill factor that is a function of the software size, and $t_d$ is the total development time measured in years.

Using the equations provided by Putnam and Myers [23, p. 234], the SLIM cost estimation equation can be written as:

$$E = 56.6\,B\,(\frac{S}{PP})^{1.29}, \tag{3}$$

where $E$ is the person-month effort, $S$ is the software size in SLOC, $B$ is the skill factor, and $PP$ is the productivity parameter.

The main advantage of the new SLIM model is that it can be easily calibrated to a software system and its production environment. The $B$ value is directly based on the size of the software to be developed, and the $PP$ reflects the productivity of the particular environment. This makes the SLIM model adaptable to changing environment and systems.

Like many other software cost models, the COCOMO and SLIM models are empirical in nature, although SLIM has its origin in the Rayleigh manpower distribution curve [22]. As a result, the models and their performances tend to vary significantly from one environment to another. Many studies (e.g., [8, 16]) have shown that these models performed poorly if their parameters were not calibrated to the particular environment in which they were to be used. It is also difficult to interpret the meaning, if any, of the parameters and the models.

The impetus for rigorous theoretical studies of software production and systems development are strengthening as the discipline and practice of software engineering become widely accepted. In recent years, some attempts have been made to develop theoretical models of software systems development from an economic production perspective. One notable study is by Banker, Datar, and Kemerer [2] in which a quadratic production function was proposed that relates the professional labor hours

incurred on a software maintenance project to the size and complexity of the project, measured in terms of function points and SLOC. Since the objective was to investigate the impact of the production environment factors (e.g., the ability of project managers, the level of previous experience with the application) on the productivity of software maintenance projects, no software cost function was developed; nor were software cost estimation models based on the production function proposed.

We believe that a software cost estimation method based on a sound theoretical foundation could have several significant advantages over empirical ones. First, we would know the correct meaning of each variable or parameter in the estimation equations so that their values could be meaningfully interpreted. Second, we would have more confidence with the equations and thus with the relationship they represent because we would know how they are derived and on what assumptions they are based. Finally, we would know what kind of cost the method attempts to estimate. For instance, the software cost when estimated by the method of economic production theory is the minimum cost obtainable for a software production under optimal conditions. In order for this estimated cost to have any real meaning, it must be assumed that one of the objectives of software project management is to maintain software production at the optimal production levels so that a minimum cost can be achieved. The next section presents a software cost model based on such economic production theory in the hope that we may have not only a better software cost model but also some insight into the fundamental behavior of software production processes.

## The MSCM Model

THE MSCM MODEL IS BASED ON THE PROPOSITION THAT SOFTWARE SYSTEMS development processes are economic productions. While software systems development usually involves multiple stages, it is premature to develop a mathematical software production model that addresses all stages of the development process concurrently without first having a model that addresses the individual stages. The model we propose here focuses on one individual stage of a multistage production process. If software production is considered as a single stage process, then this model applies to the entire process.

With these assumptions, software production can be modeled as a process in which a set of input resources at certain levels $(x_1, x_2, \ldots x_n)$ are used to produce a product of certain quantity $(S)$ over a period of time $(T)$. If $x_1$ number of people, $x_2$ amount of computer resources, . . . and $x_n$ amount of office supplies are used, then over a period of time $T$, a software system of size $S$ may be produced. In doing so, a certain cost $C$ will be incurred because not all of the resources are free.

The economic theory posits that the objective of an economic production is to maximize the profit, which is equivalent to the objective of minimizing cost while producing the desired output at a fixed level. Consider a software production process and let $y$ denote the maximum quantity of output (SLOC or FP) per unit time. Assume the process uses $n$ homogeneous production resources—for example, the human resources, capital, and the like, of quantities $(x_1, x_2, \ldots x_n)$ per unit time in order to

produce the output at the rate of $y$. Then the *production function* of this process is defined as:

(4) $$y = y(x_1, x_2, \ldots x_n).$$

The cost of producing this output at the rate $y$ by using these $x_i$ production resources at the unit prices $(p_1, p_2, \ldots p_n)$ is defined by the *cost function*:

(5) $$c = c(p_1, p_2, \ldots p_n, y).$$

Note that many different combinations of $(x_1, x_2, \ldots x_n)$ may yield the same $y$ defined in equation (4) but result in different costs. This is because, in general, not all $p_i$'s are equal and the resources are substitutable. The minimum cost of producing output at rate $y$ may be found by

minimize:

(6) $$c(p_1, p_2, \ldots p_n, y),$$

subject to:

(7) $$y - y(x_1, x_2, \ldots x_n) = 0;$$

(8) $$x_i \geq 0, \quad i = 1, 2, \ldots n.$$

If this optimization system can be solved, we get:

(9) $$c^* = c^*(x_1^*, x_2^*, \ldots x_n^*, p_1, p_2, \ldots p_n),$$

where $c^*$ is the minimum cost of production at rate $y$ by using the optimal combination of resources at rates $(x^*_1, x^*_2, \ldots x^*_n)$, given the unit prices $(p_1, p_2, \ldots p_n)$.

It is important to emphasize that the $y$ defined in equation (4) is the output quantity *per unit time* and the $c$ defined in equation (5) is the cost *per unit time*. In software production, we are interested in the total output quantity and the total cost over the time period of developing and delivering a complete software system. To find these totals, we assume the production function (4) and the cost function (5) are differentiable to second order. Let $Y$ denote the total output, and $C$ the total cost of producing $Y$, over a time period of $[0, T_d]$. From the second-order differentiability assumption, the total output is given by the Total Production Function:

(10) $$Y = \int_0^{T_d} y(x_1, x_2, \ldots x_n)\, dt,$$

Similarly, the total cost in the same period $[0, T_d]$ is given by the Total Cost Function:

(11) $$C = \int_0^{T_d} c(y, p_1, p_2, \ldots p_n)\, dt.$$

In the software production environment, $Y$ represents the software size. Let $S_d$ be the size of the software system to be developed. We are interested in finding the appropriate $(x_1, x_2, \ldots x_n)$ that satisfies equation (10) and minimizes equation (11). This minimum total cost $C$ for producing the total output $S_d$ in $T_d$ time may be found by solving the nonlinear optimization system:

(12)

minimize:

$$\int_0^{T_d} c\,(y, p_1, p_2, \ldots p_n)\, dt,$$

subject to:

(13)

$$\int_0^{T_d} y\,(x_1, x_2, \ldots x_n)\, dt - S_d = 0;$$

(14)

$$x_i \geq 0,\ i = 1, 2, \ldots n,$$

where $S_d$ is the size of the software to be developed, measured in terms of SLOC or FP, or any other appropriate metrics, and $T_d$ is the time allowed to develop this software.

This optimization system is the Minimum Software Cost Model (MSCM). Theoretically, the solution of MSCM, $C^* = (x^*_1, x^*_2, \ldots x^*_n)$, determines the optimal software production conditions under which the required software can be produced within the specified time and at minimum cost.

In order to develop a working mathematical model, it is further assumed that only human resources are to be considered. This is justified by the fact that more than 80 percent of the cost of software systems development comes from human-resource cost. The difficulties of measuring the other resources in economic terms, such as computer hardware and software, prevent us from considering all the variables that could be identified as input resources of software production. In addition, it is assumed that the software production function, with only human resources as the input, can be defined as a single factor Cobb-Douglas production function:[1]

(15)

$$y = kx^{\alpha},$$

where $k$ is the production technological level of the firm, $x$ is the input level of human resources, measured by the number of people in the production team, and $\alpha$ is the elasticity of human resources, representing the effectiveness of team members.

Theoretically, $\alpha$ can take any positive value, with $\alpha < 1$ indicating reduced team productivity, and $\alpha > 1$ indicating increased team productivity. For instance, if a software project, which could be completed by one person in ten months, is completed in four months by three people working together, then this team has an $\alpha < 1$; if it is completed by the team in three months, then the team has an $\alpha > 1$. Both scenarios are possible in the real-world software development environment. Unfortunately,

there is no analytical method for calculating the value of $\alpha$ for a given team of systems developers. The values of $k$ and $\alpha$ have to be estimated for each production environment.

With equation (15), the cost function of the software production is direct:

$$c = px. \tag{16}$$

Substituting equations (15) and (16) into the MSCM model (12, 13, and 14), it becomes the optimization problem of finding an $x$ that

minimizes

$$pxT_d, \tag{17}$$

subject to

$$kx^{\alpha} T_d - S_d = 0; \tag{18}$$

$$x > 0. \tag{19}$$

This problem can be solved using the Lagrangean method. The Lagrangean function for this problem can be written as:

$$L = pxT_d + \lambda (kx^{\alpha} T_d - S_d). \tag{20}$$

Assume an $x^*$ exists that minimizes the objective equation (17) and satisfies the constraints of (18) and (19). According to the Lagrangean theorem, the necessary conditions for the optimization problem to have a minimum at $x^*$ are:

$$L_x = \frac{\partial L}{\partial x} = pT_d + \lambda k \alpha x^{\alpha - 1} T_d = 0; \tag{21}$$

$$L_\lambda = \frac{\partial L}{\partial \lambda} = kx^{\alpha} T_d - S_d = 0. \tag{22}$$

Solving these two equations yields:

$$x^* = \left(\frac{S_d}{kT_d}\right)^{\frac{1}{\alpha}} = K\left(\frac{S_d}{T_d}\right)^{\frac{1}{\alpha}}, \tag{23}$$

where $K = (1/k)^{1/\alpha}$. Obviously $x^* > 0$ since $S_d$, $K$, and $T_d$ are positive and non-zero; therefore, constraint (19) is also satisfied. Since $x^*$ is the only solution to the Lagrangean equations, and by definition there must be a minimum in the MSCM model, this $x^*$ is indeed the solution to the MSCM model.

Substituting $x^*$ into equation (16) and then into equation (12), we get the minimum cost of the software production under the special circumstances discussed above:

$$C^* = pKS_d^{1/\alpha} T_d^{1 - \frac{1}{\alpha}}, \tag{24}$$

where $C^*$ is the minimum cost, $p$ is the unit price of human resources, $S_d$ is the size of the software system to be developed, $\alpha$ is the effectiveness of cooperation among the team members, and $T_d$ is the allowed development time.

Software cost traditionally has been measured in terms of *person-month* type of metrics. Although the person-month metric may not be appropriate for measuring software development "effort" [6, 10], it is an appropriate metric for software cost. Almost all the historical data of software cost are recorded in person-month or its variations. For this reason, we divide both sides of equation (24) by $p$ so that the cost is measured in person-months rather than monetary terms:

$$E = KS_d^{1/\alpha} T_d^{1-\frac{1}{\alpha}}. \tag{25}$$

Thus, to estimate the cost of software development with the MSCM method, the values of $T_d$, $S_d$, $K$, and $\alpha$ must be known. The method assumes that $T_d$ is given. The estimation of software size $S_d$ is by itself a complicated and difficult issue. Boehm [3] and Putnam and Myers [23] provide detailed discussions on this subject. Recently, Function Points technique has become widely accepted. Detailed discussions of the use of function points for measuring software size can be found in [9].

The parameters $\alpha$, the team cooperative effectiveness, and $K$, the production technological level, depend on the characteristics of individual software production environment. Currently, the only way to determine the values of $K$ and $\alpha$ is through estimation using historical project data. Future studies may yield some practical measurement instruments with sound theoretical basis for these two critical parameters.

## Empirical Validation

THE THREE ALGORITHMIC SOFTWARE COST MODELS, MSCM, SLIM, and COCOMO, are developed from different backgrounds and different approaches. COCOMO is an empirical model derived from statistical regression. SLIM has its roots in the Rayleigh manpower distribution. The MSCM is based on the economic production theory. When used for software cost estimation, the three models have comparable complexity in terms of the amount of computations involved. Here, however, we examine whether they have comparable quality in software cost estimation.

### Data

The software project data set of Kemerer [16] is chosen to test the performance of the MSCM model against the COCOMO and SLIM models. Several considerations are involved in this choice. First, the software projects in the Kemerer data set are all business applications and are written mostly in COBOL (except two cases) and developed in the same environment. This gives the data the necessary consistency and integrity for testing models. Second, because of the nature of the company from which the project data were collected, it seems that the data set has better accuracy and

reliability than most other public-domain data sets. Third, the data set contains mainly medium to large software projects, which are believed to exhibit relatively consistent characteristics [25]. With an average size of 221K SLOC, the Kemerer data set is superior to most of the data sets for testing the economic characteristics of software development, such as cost and time. Finally, since many previous studies have used this data set for testing various software cost models [16, 20, 26], it would be beneficial to use this data set so that the results can be compared.

Previous studies [5, 18] have noted that project nos. 3, 4, and 9 in the original Kemerer [16] data set are possible outliers. To avoid biased results resulting from the inclusion of outliers, the Mahalanobis $D^2$ distance between individual data point and the rest of the data set is calculated. Table 1 presents the result. Based on the recommendation of Hair et al. [12, p. 59] that only the observations that have large $D^2$ with the significance exceeding 0.001 be considered as outliers, project no. 3, which has an exceptionally large $D^2$ with $p < 0.000$, is determined as an outlier and excluded from the test data set. As a result, the fourteen-project data set is used to evaluate different software cost models in this study.

## Method

We chose the goodness of fit and the quality of estimation as two criteria to measure the performance of the MSCM model against other software cost estimation models. For comparative purpose, in addition to COCOMO and SLIM, two classic production models, the generalized Cobb-Douglas production function (GCD) and the generalized Cobb-Douglas production function with time factor (GCDT), are also included. The following is a list of the models to be compared:

(26) $$\text{MSCM: } E = KS_d^{\alpha} T_d^{1-\alpha}$$

(27) $$\text{GCDT: } E = KS_d^{\alpha} T_d^{\beta}$$

(28) $$\text{GCD: } E = KS_d^{\alpha}$$

(29) $$\text{COCOMO: } E = KS_d^{\alpha} \prod_{i=1}^{15} c_i$$

(30) $$\text{SLIM: } E = 56.6B\left(\frac{S_d}{PP}\right)^{1.29}$$

To be consistent with the expression of the other models, the notation of the MSCM is modified. The parameter $1/\alpha$ in equation (25) is replaced by $\alpha$ in equation (26) for mathematical parsimony. Note that the MSCM model also takes the general form of the Cobb-Douglas production function, except that it requires $\alpha + \beta = 1$. The GCDT and GCD models are included for comparison to make sure the tests will show whether or not the MSCM is a production model that better incorporates the unique character-

Table 1. The Malahanobis $D^2$ for Detecting Outliers

| Project no. | $D^2$ | $F$ | $p$ |
|---|---|---|---|
| 1 | 0.355 | 0.094 | 0.962 |
| 2 | 2.244 | 0.591 | 0.634 |
| 3 | 211.732 | 55.738 | 0.000 |
| 4 | 1.392 | 0.336 | 0.779 |
| 5 | 11.580 | 3.048 | 0.074 |
| 6 | 2.724 | 0.717 | 0.562 |
| 7 | 2.660 | 0.700 | 0.571 |
| 8 | 0.618 | 0.163 | 0.919 |
| 9 | 3.145 | 0.828 | 0.506 |
| 10 | 2.936 | 0.773 | 0.533 |
| 11 | 0.459 | 0.121 | 0.946 |
| 12 | 12.295 | 3.237 | 0.064 |
| 13 | 0.868 | 0.229 | 0.875 |
| 14 | 3.788 | 0.997 | 0.430 |
| 15 | 1.042 | 0.274 | 0.843 |

istics of software production than other general production models.

The goodness of fit of models is usually measured in terms of the $F$ statistic or the adjusted coefficient of determination, adjusted $R^2$. However, since the parameters of the Intermediate COCOMO and SLIM models are not statistically estimated, neither $F$ nor adjusted $R^2$ can be computed. Here we introduce the average square root error (ASRE) for comparing the goodness of fit of different models. It is defined as:

$$ASRE = \frac{1}{n}\sqrt{\sum_{i-1}^{n}(E_{0i} - E_{1i})^2} \tag{31}$$

where $n$ is the size of the entire data set, $E_{0i}$ is the actual cost of the $i$th project, and $E_{1i}$ is the fitted cost of the $i$th project using an estimation model. It can be seen that ASRE measures the average deviation of estimates from the actual values for the entire data set.

The difference between the ASRE and the commonly used MSE (mean square error) in statistics is that MSE uses the degree of freedom for averaging the square error terms instead of the sample size $n$ in the ASER. We did not choose MSE simply because the Intermediate COCOMO uses fifteen cost drivers, which would severely skew the result whether or not they were considered as parameters.

In computing ASRE, the entire data set was used in estimating the parameters of the cost models. In reality, however, software engineers and managers have to use the existing data set to estimate the unknown, that is, to make a predication. To test the quality of the predictions that each model makes, we used a subset of thirteen projects of the fourteen-project set to estimate the parameters of the models and then used the model to estimate the cost of the one remaining project, resulting in fourteen estimations.

Like others, we used the magnitude of error (MRE) to measure the quality of estimation of each model, as defined in [16]:

$$MRE = \left| \frac{E_1 - E_0}{E_0} \right| . \tag{32}$$

To estimate the parameters in the MSCM model, we used the least-square nonlinear regression procedure provided in the SAS statistical package. The parameters in the GCDT and GCD models were estimated using the general linear regression procedure in the SAS package by taking a logarithmic transformation. The parameter estimation for the SLIM model takes three steps: First the $B$ value for each project is determined based on the project size using Putnam and Myers's Table 2.1 [23, p. 29]. Then the $PP$ is calculated using the known values of each project using equation (2). Finally, the average of $PP$'s is compared with the $PP$ cluster values in Putnam and Myers's Table 2.3 of [23, p. 33]; the closest $PP$ cluster value in the table was chosen as the $PP$ for the SLIM model. All estimates of the Intermediate COCOMO were taken directly from [16], except the ASRE values, which were calculated using equation (31).

## Results and Discussions

WE NOW PRESENT THE RESULTS OF VARIOUS TESTS CONDUCTED using the models and data sets discussed above. First, the model parameters, if not given, were estimated. Then the ASREs were calculated for comparing the goodness of fit of the models. Finally the MREs were computed for determining the quality of estimation of the models.

### Estimation of Model Parameters

Table 2 presents the regression estimated model parameters using the fourteen-project data set; Table 3 presents the similarly estimated model parameters but with thirteen projects at a time.[2] The data set number in the table indicates the project number that was excluded from the set used for parameter estimation. The main purpose was to use the estimated parameters of the models to predict the cost of the excluded project.

In general, the models exhibit satisfactory stability in terms of the estimated values of the parameters. The means of the estimated $\alpha$, $\beta$, and $K$ with the thirteen-project data sets matched the corresponding values estimated with the full fourteen-project data set, with the exception of one $K$ estimate in the GCD model. Careful examination of the standard deviation of the estimated parameters revealed that the MSCM exhibits the most stable characteristics across all test data sets with the estimated $\alpha$ and $K$ having a standard deviation of less than 11 and 16 percent of their means, respectively. On the other hand, the estimates of $\alpha$ and $K$ of the GCDT model had a standard deviation of larger than 12 and 64 percent, respectively, and those of the GCD model were about 8 percent and 126 percent. This is one indication that the MSCM model may be a closer representation of the software systems development environment than the other models tested.

Table 2. Model Parameters Estimated with the 14-Project Data Set

| | Parameters | | |
|---|---|---|---|
| Model | $\alpha$ | $\beta$ | $K$ |
| MSCM | 0.5812 | N/A | 2.7344 |
| GCDT | 0.4784 | 0.3959 | 4.6154 |
| GCD | 0.6643 | N/A | 5.0534 |

Table 3. Model Parameters Estimated with 13-Project Data Set

| | MSCM | | GCDT | | | GCD | |
|---|---|---|---|---|---|---|---|
| Data set | $\alpha$ | $K$ | $\alpha$ | $\beta$ | $K$ | $\alpha$ | K |
| 1 | 0.5726 | 2.6886 | 0.4467 | 0.3960 | 5.2233 | 0.6326 | 5.7188 |
| 2 | 0.5898 | 2.6633 | 0.5303 | 0.3985 | 3.4594 | 0.7164 | 3.8116 |
| 4 | 0.5599 | 3.0853 | 0.5082 | 0.4417 | 3.7938 | 0.7116 | 4.2590 |
| 5 | 0.4006 | 4.0065 | 0.3494 | 0.5072 | 6.2670 | 0.6382 | 5.6576 |
| 6 | 0.5847 | 2.6987 | 0.4891 | 0.4844 | 3.3853 | 0.6899 | 4.3837 |
| 7 | 0.5779 | 2.7659 | 0.4238 | 0.2364 | 1.6308 | 0.5216 | 10.9894 |
| 8 | 0.5907 | 2.7028 | 0.4962 | 0.3759 | 1.6231 | 0.6795 | 4.7968 |
| 9 | 0.6453 | 2.4657 | 0.5662 | 0.3347 | 1.2667 | 0.7324 | 3.8358 |
| 10 | 0.5856 | 2.6949 | 0.5122 | 0.4583 | 1.4563 | 0.7019 | 4.1255 |
| 11 | 0.5508 | 2.8655 | 0.4244 | 0.4398 | 1.3975 | 0.6414 | 2.0176 |
| 12 | 0.6714 | 2.0912 | 0.5422 | 0.2594 | 1.5814 | 0.6639 | 1.8992 |
| 13 | 0.5631 | 2.9067 | 0.4743 | 0.4104 | 1.5525 | 0.6624 | 1.9423 |
| 14 | 0.6356 | 2.3067 | 0.5054 | 0.3183 | 1.2962 | 0.6485 | 1.9394 |
| 15 | 0.5678 | 2.8445 | 0.4178 | 0.4625 | 1.5074 | 0.6557 | 1.9127 |
| $\mu$ | 0.5783 | 2.7705 | 0.4776 | 0.3945 | 2.5315 | 0.6640 | 1.9264 |
| $\sigma$ | 0.0619 | 0.4353 | 0.0588 | 0.0818 | 1.6290 | 0.0517 | 2.4265 |

## Goodness of Fit

Table 4 shows the result of testing the goodness of fit using ASRE. The $F$ statistics and the adjusted $R^2$ values are also presented where they were available. Judged by the ASRE values, the MSCM model had the best overall fit to the data set, and the Intermediate COCOMO had the worst fit, with GCDT, GCD, and SLIM in the middle. The adjusted $R^2$ values were consistent with the ASREs for MSCM, GCDT, and GCD models. One may argue that the four better models all use the parameters estimated using the data set, while the Intermediate COCOMO uses constant parameters provided by Boehm [8]. But the Intermediate COCOMO estimates have been adjusted by fifteen cost drivers reflecting far more detailed characteristics of the projects and the development environment. Thus, we believe the results in Table 4 show a fair comparison of these five models.

Table 4. The Goodness of Fit of Models

| | Model | | | | |
|---|---|---|---|---|---|
| | MSCM | GCDT | GCD | COCOMO | SLIM |
| ASRE | 15.13 | 15.91 | 16.92 | 339.49 | 21.67 |
| $F$ | 55.81 | 9.44 | 16.08 | N/A | N/A |
| $R^2$ adj. | 0.89 | 0.56 | 0.54 | N/A | N/A |

## Quality of Estimation

The best-fitting model may not necessarily have the best quality when used for estimating the cost of future projects. Overfitting to a data set may result in biased models, as shown by [18], producing poor estimates. To compare the estimation quality of the five models, actual data of thirteen of the fourteen projects were used for parameter estimation, and the one remaining project was used for testing. In total, fourteen data sets were constructed and fourteen estimations were made by each model. MREs for each estimate were calculated to evaluate the estimation quality. Table 5 presents the results.[3]

In terms of mean MRE, the MSCM model had the best overall estimation quality, as indicated by the 50 percent MRE, followed closely by the SLIM model with a 53 percent MRE; the Intermediate COCOMO was the worst of all, with a 593 percent MRE, more than ten times larger than those of MSCM and SLIM. If we examine the standard deviation of the MREs, then the SLIM took the lead with 28 percent, indicating that it was more stable from project to project. This may be attributed to the fact that it used a single *PP* value for all projects in the same environment. The MSCM took the third position with 42 percent; the Intermediate COCOMO once again was the worst of all, with 895 percent, more than twenty times larger than the SLIM and MSCM models. When judged by the criteria of percentage of estimates having an MRE less than or equal to 25 percent (25%-MRE), 30 percent (30%-MRE), or 50 percent (50%-MRE), as shown in Table 6, then the MSCM and the SLIM models gave comparable performance, with MSCM taking an overall lead: The SLIM led in the 25%-MRE category by 7 percent (one project) over the MSCM, while the MSCM led in the 30%-MRE and 50%-MRE categories over the SLIM by 14 percent (two projects) and 36 percent (five projects). In all cases, the Intermediate COCOMO was no match for either MSCM or SLIM: It produced no estimate that fell into any of the categories.

It should be pointed out that the MSCM estimation used the actual production time ($T_d$). This may or may not be a problem for real-world applications. In many cases, software development teams are given deadlines for delivering specific software systems. The cost of developing such software can be estimated by using this schedule and the estimated software size. It is possible, though unlikely, that there would be no specific timetable for a system development project. In this situation, the MSCM can be used to estimate the costs of development under different schedule scenarios for

Table 5. The Quality of Estimation

| | | MSCM | | GCDT | | GCD | | COCOMO | | SLIM | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Project no. | Actual | Estimate | MRE (%) | Estimate | MRE (%) | Estimate | MRE (%) | Estimate | MRE(%) | Estimate | MRE(%) |
| 1 | 287.00 | 214.80 | 25 | 190.10 | 34 | 189.77 | 34 | 917.56 | 220 | 197.60 | 31 |
| 2 | 82.50 | 52.50 | 36 | 53.48 | 35 | 54.04 | 35 | 151.66 | 84 | 16.16 | 80 |
| 4 | 86.90 | 222.33 | 156 | 208.12 | 139 | 194.23 | 124 | 558.98 | 543 | 159.12 | 83 |
| 5 | 336.30 | 215.43 | 36 | 194.56 | 42 | 279.08 | 17 | 1,344.2 | 300 | 413.96 | 23 |
| 6 | 84.00 | 51.86 | 38 | 50.01 | 40 | 65.17 | 22 | 313.36 | 273 | 23.08 | 73 |
| 7 | 23.20 | 47.96 | 107 | 11.75 | 49 | 78.16 | 237 | 234.78 | 912 | 17.46 | 25 |
| 8 | 130.30 | 164.92 | 27 | 55.43 | 57 | 175.58 | 35 | 1,165.7 | 795 | 145.47 | 12 |
| 9 | 116.00 | 243.49 | 110 | 75.78 | 35 | 243.28 | 110 | 4,248.73 | 3,563 | 233.88 | 102 |
| 10 | 72.00 | 44.87 | 38 | 19.88 | 72 | 53.98 | 25 | 180.29 | 150 | 15.39 | 79 |
| 11 | 258.70 | 191.58 | 26 | 45.30 | 82 | 70.40 | 73 | 1,520.04 | 488 | 198.21 | 23 |
| 12 | 230.70 | 168.50 | 27 | 53.65 | 77 | 47.74 | 79 | 558.12 | 142 | 82.29 | 64 |
| 13 | 157.00 | 188.40 | 20 | 59.17 | 62 | 56.34 | 64 | 1,073.47 | 584 | 110.32 | 30 |
| 14 | 246.90 | 193.96 | 21 | 48.25 | 80 | 53.13 | 78 | 629.22 | 155 | 113.32 | 54 |
| 15 | 69.90 | 91.16 | 30 | 28.30 | 60 | 28.08 | 60 | 133.94 | 92 | 30.12 | 57 |
| Max | | | 156 | | 139 | | 237 | | 3,563 | | 102 |
| Min | | | 20 | | 34 | | 17 | | 84 | | 12 |
| $\mu$ | | | 50 | | 62 | | 71 | | 593 | | 53 |
| $\sigma$ | | | 42 | | 28 | | 58 | | 895 | | 28 |

Table 6. Estimation Quality Measured in Percentage of Estimates in Each Error Category

| | Model | | | | |
|---|---|---|---|---|---|
| | MSCM | GCDT | GCD | COCOMO | SLIM |
| 25%-MRE | 21 | 0 | 21 | 0 | 28 |
| 30%-MRE | 50 | 0 | 21 | 0 | 36 |
| 50%-MRE | 79 | 43 | 43 | 0 | 43 |

evaluation and decision-making purposes. Comparing with the single schedule estimate provided in the COCOMO and SLIM methods, we consider the MSCM approach more realistic and more flexible, and especially valuable for conducting what-if analyses for project planning.

## Implications of the MSCM Model

THE TEST RESULTS PRESENTED ABOVE VERIFY THE CAPABILITY of the MSCM model as a software cost model. Since it is based on the foundation of economic production theory, the model could provide many meaningful implications for software project management policies and practices. First, the estimated $\alpha = 0.5812$ for equation (26) of the MSCM model translates into an 1.72 effectiveness of cooperation of the development teams as defined in equations (15) and (25), indicating that in this production environment the team productivity is higher than the individual. This may have been a result of better system design that allowed modular divisions of the tasks among team members and effective team management practices.

Second, equation (26) of the MSCM model implies that there is a tradeoff between the cost ($E$) and development schedule ($T_d$), other factors being equal. This relationship can be examined by the partial derivative of $E$ with respect to $T_d$:

(33)
$$\frac{\partial E}{\partial T_d} = (1-\alpha) K \left(\frac{S_d}{T_d}\right)^{\alpha}.$$

Since $(1-\alpha)$ and $K$ are greater than zero, so $\partial E/\partial T_d > 0$, indicating that the cost of a software project may be reduced by tightening its development schedule, which requires increasing the team size as dictated by equation (23). This is consistent with the first implication about team productivity in this particular software development environment. Equation (33) also implies that as the development schedule gets longer, its effect on the effort diminishes. Figure 1 illustrates this tradeoff between the cost ($E$) and the schedule ($T_d$).

Finally, the effect of software size ($S_d$) on development cost ($E$) can be shown by the partial derivative of $E$ with respect to $S_d$:

(34)
$$\frac{\partial E}{\partial S_d} = K \alpha \left(\frac{T_d}{S_d}\right)^{1-\alpha}.$$

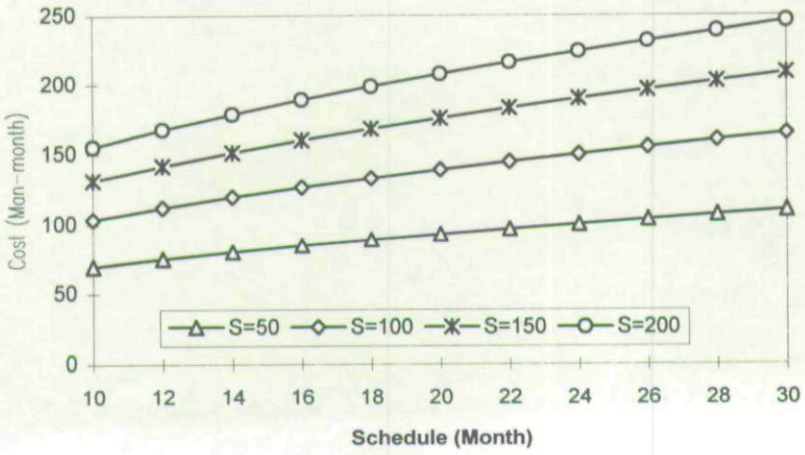


*Figure 1.* Tradeoff between Cost and Schedule

Since $\alpha$ and $K$ are greater than zero, so $\partial E/\partial S_d > 0$, indicating that the cost of a software project always increases as the size of software increases. However, equation (34) also suggests that when software size gets very large, the effect of size increase on the cost diminishes, or, in other words, the marginal cost per line of code gets smaller, implying that larger software costs less than smaller software in terms of person-month/SLOC in this particular environment. This is clearly shown in figure 2, which is plotted using the actual productivity data provided in [16], except that project no. 3 is excluded as an outlier. This contrasts to the MPSS (most productive scale size) concept of Banker and Kemerer [1] who hypothesized that there were increasing returns to scale for smaller projects and decreasing returns for very large projects.

However, caution should be exercised when considering these implications. The conclusions of partial derivative analyses are valid only if the changes of the independent variables are relatively small. A significantly reduced development schedule and increased team size, for instance, may change production characteristics which can eventually push the effectiveness of cooperation into the region of less than one. That would result in very different interpretations. In addition, the underlying assumption of partial derivative analysis is that other factors must be kept unchanged when the independent variable changes, which in reality may be difficult to implement. For instance, it may be difficult to keep the effectiveness of cooperation of team members unchanged while the size of software is being increased or the production schedule is being tightened. The value of such analysis, though, is to point out the correct directions for changes if they become necessary, and to warn off possible adverse economic affects if certain factors have to be adjusted because of technical or political causes.

## Conclusions

THE PURSUIT OF BETTER SOFTWARE COST MODELS MAY BE A NEVER-ENDING task for software engineering researchers and practitioners in view of the ever-changing

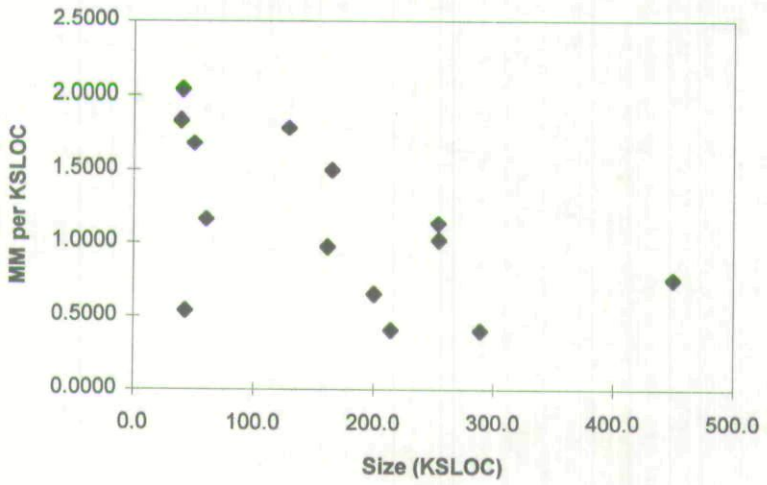


*Figure 2.* Marginal Cost of Software Project

software applications and software systems development technologies. In order to estimate the dynamic, sometimes even chaotic, behavior of software systems development cost with any degree of certainty, we must develop software production models based on solid theoretical foundations. This study has shown how the basic approach of economic production theory may be applied to develop better software cost models. Starting with the generic Cobb-Douglas production function and incorporating the unique requirement of software cost estimation, we developed a software cost model called MSCM. Using the software project data set of Kemerer, we compared the performance of the MSCM model with the classic Intermediate COCOMO and the newly revised SLIM in terms of goodness of fit to the data set and the quality of estimation. In either category, the MSCM's performance was far better than the COCOMO and comparable to, if not better than, the new SLIM model.

In addition to being a capable cost estimation model, the MSCM also offers interesting implications about the characteristics of the software production environment and the relationships among the important variables. By examining the value of estimated parameter $\alpha$, the MSCM model suggests that the development teams had higher productivity than individuals; thus, using a larger team may shorten the production schedule and, to a certain extent, reduce the software cost, other factors being equal. The MSCM also implies that, in this particular environment, the team may build larger software systems more cheaply than smaller ones in terms of cost per line of code.

The generalization of the conclusions of this study, however, may be limited by the size of the software project data set on the basis of which the parameters of the MSCM model were estimated and different models compared. Although other larger historical data sets are available in the literature, the age of these data sets, the uncertain quality of the data, and the lack of information on homogeneity of the projects and development environment, would have severe effects on any conclusions drawn. It is our hope that this study may inspire more practitioners to use the MSCM model in their software

development environment in addition to whatever they have been using for software cost estimation so that the validity and the accuracy of this theoretically derived model may be further tested and verified. We also call for researchers and practitioners in software engineering field to publish their data sets when comparing software cost models so that different models can be fairly compared and new and better models can be developed. In addition to more empirical testing, future studies should focus on developing practical instruments for measuring the values of $\alpha$ and $K$ for any given software production environment. The use of the Cobb-Douglas production function as the starting point for the development of the MSCM model should also be subjected to more theoretical and empirical scrutiny.

*Acknowledgment*: We thank the three anonymous referees for their insightful and valuable comments.

## NOTES

1. The choice of the Cobb-Douglas function is based mainly on two factors: First, the Cobb-Douglas function is the most widely used and tested economic production function in the literature for mathematical parsimony and power; second, the more powerful translog production function, which is considered more generic than the Cobb-Douglas and provides a valid second-order approximation to an arbitrary function form [7], collapses into the Cobb-Douglas function when only one production factor is considered, as is the case here.

2. See the appendix for model parameters estimated using the original fifteen-project data set, together with a discussion of the effect of the outlier on the models.

3. See the appendix for comparison of the models using the original fifteen-project data set, together with a discussion of the effect of the outlier on the estimates.

## REFERENCES

1. Banker, R.D.; Datar, S.M.; and Kemerer, C.F. A model to evaluate variables impacting the productivity of software maintenance projects. *Management Science, 37*, 1 (1991), 1–18.

2. Banker, R.D., and Kemerer, C.F. Scale economies in new software development. *IEEE Transactions on Software Engineering, 15*, 10 (1989), 1199–1205.

3. Boehm, B.W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

4. Boehm, B.W. Improving software productivity. *Computer, 20*, 9 (1987), 43–57.

5. Briand, L.C.; Basili, V.R.; and Thomas, W.M. A pattern recognition approach for software engineering data analysis. *IEEE Transactions on Software Engineering, 18*, 11 (1992), 931–942.

6. Brooks, F.P. *The Mythical Man-Month.* Reading, MA: Addison-Wesley, 1975.

7. Christenson, L.R.; Jorgenson, D.W.; and Lau, L.J. Conjugate duality and the transcendental logarithmic function. *Econometrica, 39*, 4 (1971), 255–256.

8. Cuelenaere, A.; van Genuchten, M.; and Heemstra, F. Calibrating a software cost estimation model: why and how. *Information and Software Technology, 29*, 10 (1987), 558–567.

9. Dreger, J.B. *Function Point Analysis*. Englewood Cliffs, NJ: Prentice-Hall, 1989.

10. Ejiogu, L.O. *Software Engineering with Formal Metrics*. Wellesley, MA: QED Technical Publishing Group, 1991.

11. van Genuchten, M. *Towards a Software Factory.* Dordrecht, the Netherlands: Kluwer Academic Publishers, 1992.

12. Hair, J.F., Jr.; Anderson, R.E.; Tatham, R.L.; and Black, W.C. *Multivariate Data Analysis*, 4th ed. Englewood Cliffs, NJ: Prentice-Hall, 1995.

13. Jenkins, A.M.; Naumann, J.D.; and Wetherbe, J.C. Empirical investigation of systems development practices and results. *Information and Management*, *7*, 2 (1984), 72–83.

14. Jorgensen, M. Experience with the accuracy of software maintenance task effort prediction models. *IEEE Transactions on Software Engineering*, *21*, 8 (1995), 674–681.

15. Keil, M.; Mixon, R.; Saarinen, T.; and Tuunaiene, V. Understanding runaway information technology projects: results from an international research program based on escalation theory. *Journal of Management Information Systems*, *11*, 3 (1995), 65–85.

16. Kemerer, C.E. An empirical validation of software cost estimation models. *Communications of the ACM*, *30*, 5 (1987), 416–429.

17. Kitchenham, B.A., and Taylor, N.R. Software project cost estimation. *Journal of Systems and Software*, *5*, 5 (1985), 267–278.

18. Matson, J.E.; Barrett, B.E.; and Mellichamp, J.M. Software development cost estimation using function points. *IEEE Transactions on Software Engineering*, *20*, 4 (1994), 275–287.

19. Mohanty, S.N. Software cost estimation: present and future. *Software—Practice and Experience*, *11*, 2 (1981), 103–121.

20. Mukhopadhyay, T.; Vicinanza, S.S.; and Prietula, M.J. Examining the feasibility of a case-based reasoning model for software effort estimation. *MIS Quarterly*, *16*, 2 (1992), 155–171.

21. Nidumolu, S. The effect of coordination and uncertainty on software project performance: residual performance risk as an intervening variable. *Information Systems Research*, *6*, 3 (1995), 191–219.

22. Putnam, L.H. A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering*, *SE–4*, 4 (1978), 345–361.

23. Putnam, L.H., and Myers, W. *Measures for Excellence: Reliable Software on Time, within Budget*. Englewood Cliffs, NJ: Yourdon Press, 1992.

24. Rothfeder, J. It's late, costly, incompetent—but try firing a computer system. *Business Week* (November 7, 1988), 164–165.

25. Scacchi, W. Understanding software productivity: towards a knowledge-based approach. *International Journal of Software Engineering and Knowledge Engineering*, *1*, 3 (1991) 293–321.

26. Srinivasan, K., and Fisher, D. Machine learning approaches to estimating software development effort. *IEEE Transactions on Software Engineering*, *21*, 2 (1995), 126–137.

27. Vicinanza, S.S.; Mukhopadhyay, T.; and Prietula, M.J. Software effort estimation: an exploratory study of expert performance. *Information Systems Research*, *2*, 4 (1991), 243–262.

28. Zmud, R.W. Management of large software development efforts. *MIS Quarterly*, *4*, 2 (1980), 45–55.

## APPENDIX

THIS APPENDIX DISCUSSES THE EFFECT OF THE OUTLIER, project no. 3 in Kemerer's original data set [16], on the estimated model parameters. We also present a comparison of the estimation quality of the MSCM model with three other models used in the study by Mukhopadhyay, Vicinanza, and Prietula [20].

Table 7 shows the estimated model parameters using the full data set (fourteen projects at a time), including the outlier. The values can be contrasted with those in Table 3. The inclusion of the outlier generally increased the variation of the estimated parameters in all models. The most significant effect is on the parameters of the MSCM model, as shown by the changes of the $K$ value (i.e., from 2.7705 to 0.7082). This can be explained by the structures of the models. In the MSCM model, the parameter $K$ is defined in equation (23) as $(1/k)^{1/\alpha}$, where $k$ is the technological level, and $\alpha$ is the cooperative efficiency of team members. Thus, a small change in $\alpha$ results in a

Table 7. Estimated Model Parameters with the Data Sets Containing the Outlier

| | MSCM | | GCDT | | | GCD | |
|---|---|---|---|---|---|---|---|
| | $\alpha$ | $K$ | $\alpha$ | $\beta$ | $K$ | $\alpha$ | $K$ |
| $\mu$ | 0.7612 | 0.7082 | 0.7144 | 0.2276 | 1.7966 | 0.8118 | 2.2416 |
| $\sigma$ | 0.1350 | 0.5946 | 0.0758 | 0.0896 | 0.9126 | 0.0600 | 1.0186 |

Table 8. Comparison of Quality of Estimation of Different Models: MRE (%)

| | MSCM | GCDT | GCD | COCOMO | SLIM | Expert | ESTOR | Function point |
|---|---|---|---|---|---|---|---|---|
| Max | 139 | 117 | 139 | 3,563 | 102 | 72.41 | 106.9 | 326.72 |
| Min | 2 | 12 | 7 | 84 | 12 | 0.86 | 0.98 | 0.23 |
| $\mu$ | 67 | 52 | 52 | 584 | 53 | 30.72 | 52.79 | 102.74 |
| $\sigma$ | 33 | 29 | 43 | 863 | 28 | 21.74 | 37.92 | 116.05 |

significant change in $K$, while in GCDT and GCD, $K$ and $\alpha$ are independent; hence the effect of the outlier is not magnified as in the MSCM model.

Table 8 compares the quality of estimation of the models presented in this study with three models in the study by Mukhopadhyay, Vicinanza, and Prietula [20]. It is not surprising that the Experts produced the best estimation. The case-based expert system Estor, the two production function models, the GCDT and GCD, as well as the SLIM model have comparable performance. The MSCM model is much better than the Function Point model and the Intermediate COCOMO, but not as good as the Estor model. However, caution should be exercised when interpreting these results. First, these quality indicators are derived using the full data set, which contains the identified outlier that may skew the results of different models to a differing extent. Second, the quality indicators of the Estor model are based on the verbal protocols of human expert solving the first ten projects [20] that overlap with the test data set, while those of our models have no overlap cases in the tests. Table 7 is provided only for reference purposes.