Advances in
COMPUTERS

EDITED BY

MARSHALL C. YOVITS

VOLUME 24

Advances in
COMPUTERS

EDITED BY

MARSHALL C. YOVITS

Purdue School of Science
Indiana University—Purdue University at Indianapolis
Indianapolis, Indiana

VOLUME 24

1985

ACADEMIC PRESS, INC.

(Harcourt Brace Jovanovich, Publishers)

Orlando ■ San Diego ■ New York ■ London
Toronto ■ Montreal ■ Sydney ■ Tokyo

ACADEMIC PRESS, INC.
Orlando, Florida 32887

United Kingdom Edition published by
ACADEMIC PRESS INC. (LONDON) LTD.
24–28 Oval Road, London NW1 7DX

LIBRARY OF CONGRESS CATALOG CARD NUMBER:

ISBN: 0-12-012124-7

PRINTED IN THE UNITED STATES OF AMERICA

85 86 87 88 9 8 7 6 5 4 3 2 1

Preface

The publication of Volume 24 of *Advances in Computers* continues the presentation in depth of subjects of both current and long-range interest to the computer and information science community. Contributions have been solicited from widely known experts in their fields who have recognized the significance of writing substantial review and tutorial articles in their areas of expertise. *Advances in Computers* permits the publication of survey-type articles which have been written from a relatively leisurely perspective, and the subjects of these articles are treated both in depth and in breadth. *Advances in Computers* is a series that began in 1960 and now continues in its 25th year with Volume 24. During this period, which witnessed great expansion and dynamic change in the computer and information fields, this series has played an important role in the development of computers and their applications. The continuation of this series over such a long period of time is a tribute to the quality of presentation and to the reputations and capabilities of the authors who have written articles for the series.

Included in Volume 24 are articles on software productivity, operating systems, microprogramming (or firmware), pattern recognition and learning, language data processing, information retrieval, and computer science education, all topics of substantial and growing current and long-range interest.

In the first article Conte, Dunsmore, and Shen discuss the problems of estimating software effort and cost, a critical problem faced by all software project managers, and they remind us that this problem is important for many different reasons. In their article they discuss and evaluate several models for software effort estimation and they attempt to compare the performance of these models on sets of projects for which some information is available. They conclude that further experimentation, the gathering of more data, and the combining and enhancing of models will be necessary in order to allow computer scientists to explain and better control the software development process.

Dr. Michael Harrison considers the problem of data protection in operating systems. He points out that modern computer systems contain important information and that unauthorized access can result in significant problems. He references the IRS, electronic funds transfer, and military systems as examples. Dr. Harrison treats this problem theoretically from an operating systems point of view, and shows that there are certain inescapable conclusions which are not unlike the problems that one faces with respect to

physical security. There is constant tension between the abilities of a potential criminal and the countermeasures that can be taken. He concludes finally that the best simple technique that one can utilize is some form of encryption method. Furthermore, he states that as time goes by it will be ever more essential to provide some sort of security mechanisms for commercial systems.

Dasgupta and Shriver point out in their article that the use of microprogramming for implementing control units and emulating instruction sets has become commonplace and in fact has even become a topic of interest to systems and applications programmers as well as to computer architects and engineers. To emphasize the user interface of microprogramming, the term "firmware" has been coined as a synonym. They trace the key developments in firmware engineering and convey some of its intellectual content. In the past ten years firmware engineering has emerged as a systematic discipline and its central problems have been essentially solved. The issue of formal verification, however, still remains controversial and yet it is of great importance in reasoning about the correctness of designs.

In his article on learning and pattern recognition Dr. Ranan Banerji points out that the word "learning," even in a limited sense, covers a large number of phenomena. Dr. Banerji discusses the phenomenon whereby someone endowed with the ability to perform a task in an inefficient manner can improve efficiency through experience; his discussion is motivated by the desire to write computer programs that have this capability. Banerji studies the problem in a way that has been used previously by workers in the field of pattern recognition, and in doing so he defines pattern recognition quite broadly. He concludes that the study of learning has been directed to specific tasks and accordingly many basic problems have been clarified. As this understanding deepens, the field, likened to artificial intelligence, will develop into two branches, applied and pure.

Professor Paul Garvin presents a survey of the field of language data processing, which is defined to mean the processing of natural language by computer. His article is a critical examination of the problem areas rather than a thorough coverage of the literature. Furthermore, he points out that much of the research which goes on in this field is subsumed under other titles such as artificial intelligence. Dr. Garvin states that the field began about 30 years ago with great enthusiasm and promise. However, in the interim a number of significant problems have been recognized. Nevertheless a continuing interest in language data processing has existed over the years. Professor Garvin still finds language data processing in general and machine translation in particular the most important intellectual challenge presented to linguistics, and although not all of his colleagues will agree with him, he believes that this has been a challenge which American linguistics

has on the whole failed to meet. He concludes that it is difficult to expect anything but rather dim prospects for a significant advance.

In his article Dr. Donald Kraft points out that information retrieval is an important part of computer science. Much effort goes into determining where to store information and how to retrieve it efficiently and effectively, and although information retrieval is not a new idea it incorporates much of the newest work in computing. The article treats information retrieval systems as a special class of information systems along with database management and question-answering systems. The key element that distinguishes information retrieval systems is the concept that the relevance of any given record to any specific query cannot be determined exactly. In recent years many commercialized systems and theoretical approaches have emerged. He concludes that information retrieval is nevertheless an interesting and challenging area of study and application, and it is an area currently in great flux.

The final article by Professor William Atchison gives a brief development of computer science education. He begins by giving several examples of the use of early computers and research projects and points out that it was from these research projects that courses, curricula, and degree programs arose. Most of the early computer courses and degrees were given in graduate schools, and, as curricula proliferated, undergraduate programs began to develop. Some of the trends in computer science education are examined which are important at the present time. For example, in early curricula, courses in programming, switching theory, and numerical analysis were emphasized. Today, in comparison, most major universities offer 50–60 different courses. Current activities in computer research will have a profound effect on the future of computer science education. Dr. Atchison also points out that since computers now affect nearly every aspect of society, the major social, ethical, philosophical, political, and legal issues related to computers will be incorporated in an increasing number of curricula.

I am pleased to thank the contributors to this volume. They have given extensively of their time and effort in order to provide a significant contribution to their profession. Despite the many calls upon their time and their busy schedules they recognize the necessity of writing substantial review and tutorial articles in their areas of expertise. Their cooperation and assistance have been greatly appreciated. Because of the efforts of these outstanding authors, this volume continues to display the high quality typical of this series. This volume should be of great value and substantial interest for many years to come. It has been a pleasant and a rewarding experience for me to edit this volume and work with these authors.

MARSHALL C. YOVITS

Contents

The Logic of Learning: A Basis for Pattern Recognition and for Improvement of Performance

Ranan B. Banerji

The Current State of Language Data Processing

Paul L. Garvin

Advances in Information Retrieval: Where Is That /#*&@¢ Record?

Donald H. Kraft

The Development of Computer Science Education

William F. Atchison

Software Effort Estimation and Productivity

S. D. CONTE, H. E. DUNSMORE, AND V. Y. SHEN

Department of Computer Science
Purdue University
West Lafayette, Indiana

1. Introduction

A critical problem faced by all software project managers is that of accurate effort and cost estimation. This is obviously true for all projects subject to competitive bidding. If the estimated bid cost is too high compared to other bids, the contract will be lost, assuming all other factors are equal. On the other hand, a winning bid which is too low may result in a loss to the organization. Accurate estimation is therefore critical to all projects of this type.

ISBN 0-12-012124-7

Even if the proposed project does not involve competitive bidding and is only for internal use in an organization, realistic cost estimation is still very important. A decision by management whether to proceed with a project may very well be based on the project leader's estimated cost, even though other factors may be taken into account. The credibility of the project manager is also at stake, especially if software costs are consistently underestimated. Even decisions as to whether a project should be done in-house or contracted out are likely to be affected by cost and effort estimates.

We have been using cost and effort almost interchangeably in the preceding discussion, and while they are closely related, cost is not necessarily a simple function of effort. Indeed, effort is usually measured in man-months[1] of the technical staff required to complete a project. Technical staff will normally include programmers, analysts, project leaders, and management directly associated with the project. A cost model must, of course, convert the technical effort estimate into a dollar cost figure, usually by computing an average salary per unit time of the technical staff involved and multiplying this average salary by the estimated effort required. It is common to use a *burdened* average salary in carrying out this computation; a burdened salary might, for example, include fringe benefits, some types of overhead, and clerical support. The means of computing the burdened average salary will vary with the organization. A cost model must also consider other direct charges in arriving at a final cost estimate; these charges might include computer time, travel costs, and general and administrative expenses.

It is evident that burdened cost models must take into account many factors which are primarily environment dependent and are therefore not easily transportable from one organization to another. Many organizations have developed in-house cost estimation models. Of particular note are models developed by TRW (Wolverton, 1974) and Boeing (Black *et al.*, 1977). In this article we will be primarily concerned with effort and cost estimation models which are appropriate for software project development. Project development will be meant to include life cycle phases from project design, through system integration, to testing and software delivery. Therefore, these models exclude the requirements analysis and specification phases, as well as postdelivery maintenance. There is some hope that effort and cost models so restricted can be developed which are transportable from one organization to another.

In recent years, a large number of effort and cost models have been proposed (see, for example, Mohanty, 1981; Section 29.7 of Boehm, 1981; Boehm, 1984). Unfortunately, for one project or for similar projects, these

[1] Some may prefer "programmer-month" or "person-month," which accurately reflects the fact that many "man-months" are the result of work by women staff members. We intend "man-month" as a sexless term.

models generally yield substantially different estimates of cost and/or effort, and while we would expect some differences because of the imprecise nature of most models of physical processes, the differences are too great to inspire much confidence in their general usefulness. For example, Mohanty (1981) compared some 20 cost models using a hypothetical software system, which consisted of some 36,000 executable machine language instructions. Insofar as possible, all of the data supplied about the system were the same for all models, including a burdened cost per man-year of $50,000. The estimated costs of the various models ranged from a low of $362,500 to a high of $2,766,667, nearly an order of magnitude difference.

In the remainder of this article we will review and, wherever possible, critically evaluate a number of cost and effort models. We are actually more interested in the methodology than in the specific models. Hence, we will group models, insofar as possible, into categories depending on the method used in deriving the model. We shall use four categories: (1) historical–experiential models, (2) statistically based models, (3) theoretically based models, and (4) composite models. Before discussing specific models, it is useful to develop criteria for judging the goodness of a model. The following are considered useful criteria:

1. *Validity.* Does the model give reasonably close estimates at least on the validating database? Are measures of confidence given for the estimates? Is the model applicable to the project under consideration?
2. *Objectivity.* Are the estimates based on measurements and data that are reproducible? Do they depend on subjective factors that can vary significantly with different human estimators?
3. *Ease of Use.* Are the data needed for the model easy to obtain? Are too many data needed? Does the effort necessary to gather the needed data require an unacceptable amount of overhead? Is the information needed available early in the life cycle?
4. *Sensitivity.* Does just a small change in one or more input parameters lead to a relatively greater change in the model estimate?
5. *Transportability.* Is the model so dependent on local data that it cannot be used in a different environment?

To aid the reader, Table I lists all the metric identifiers used in the remainder of this article and a brief description of each. As much as possible we have chosen one identifier for a metric and used it throughout. For example, even though several authors have used several symbols for "programmer effort," we have selected the most common E and use that when discussing any model. This means that some of the formulas appearing in this article are cosmetically different from their representation in the referenced work. But,

TABLE I

NOTATION USED FOR METRIC IDENTIFIERS

C	Cost matrix (Wolverton); technology factor (Putnam); communication paths (Thebaut)
$\hat{C}$	Estimated technology factor
D	Difficulty metric (Putnam; $D = K/T^2$)
D	Difficulty metric (Halstead; $D = \eta_1/2 \times N_2/\eta_2$)
E	Actual development effort in programmer-months
$\hat{E}$	Predicted (estimated) development effort
E_p	Programming effort
E_c	Coordinating effort
EC_i	Effort complexity class
I	Productivity index
K	Total life cycle effort (Putnam)
$\hat{K}$	Predicted (estimated) total life cycle effort
L	Productivity in LOC/programmer-month
$\overline{L}$	Average productivity for multimodule programs
L_{group}	Group productivity
MRE	Magnitude of the relative error ($\equiv$|RE|)
$\overline{\text{MRE}}$	Mean magnitude of the relative error
η_1	Unique operator metric
η_2	Unique operand metric
η	Vocabulary metric ($\eta = \eta_1 + \eta_2$)
N_1	Total operator metric
N_2	Total operand metric
N	Program size metric ($N = N_1 + N_2$)
P	Number of programmers on a team
$\overline{P}$	Average number of programmers on a team
PRED(r)	Prediction at level r
RE	Relative error
$\overline{\text{RE}}$	Mean relative error
RMS	Root mean square error
$\overline{\text{RMS}}$	Relative mean square error
S	Program size in thousands of lines of code (KLOC) (general); Stroud number (software science)
S_s	Program size in lines of code (LOC)
T	Time for development
$\hat{T}$	Predicted time for development
V	Volume metric
a, b, c, d	Constants
$m(X)$	Effort adjustment multiplier
n, i, j, k, m	Indices
x	Independent variable
y	Dependent variable

we believe the added clarity and aid to comprehension outweigh this minor disadvantage. Note that in a few cases one identifier (e.g., C) is used for more than one metric. In those cases context should make it clear which is intended.

2. Measures of Model Goodness

There are many measures which have been used to compare the performance of different models (see, for example, Thebaut and Shen, 1984). All of them have limitations and strengths. The following measures have been found to be most useful in our work. In this section we denote the actual effort expended on a software project by E and the effort predicted by a model by $\hat{E}$.

2.1 The Relative Error (RE) and Mean Relative Error ($\overline{\text{RE}}$)

We define the relative error by

$$\text{RE} = (E - \hat{E})/E \tag{2.1}$$

The measure RE can be negative or positive. If $\hat{E} > E$ then $\text{RE} < 0$, while if $\hat{E} < E$ then $\text{RE} > 0$. Note that when $\text{RE} > 0$ it must be between 0 and 1, while when $\text{RE} < 0$ it is essentially unbounded in magnitude. We can also define the mean relative error of a set of n projects by the formula

$$\overline{\text{RE}} = \frac{1}{n}\sum_{i=1}^{n} \text{RE}_i \tag{2.2}$$

If a model is a good representation of effort expenditure, then it will lead to small values of RE and generally to a small $\overline{\text{RE}}$. However, since it is possible that large positive REs can be balanced by large negative REs, a small $\overline{\text{RE}}$ may not imply that a model is a good one. Hence, this measure is not too useful in practice.

2.2 The Magnitude and the Mean Magnitude of the Relative Error

In view of the problem with RE and $\overline{\text{RE}}$ discussed in the preceding section, we define

$$\text{MRE} = |\text{RE}| = |(E - \hat{E})/E| \tag{2.3}$$

Thus, the smaller the value of MRE, the better the prediction. For a set of n projects we can compute the mean magnitude of the relative error by

$$\overline{\text{MRE}} = \frac{1}{n}\sum_{i=1}^{n} \text{MRE}_i \tag{2.4}$$

If $\overline{\text{MRE}}$ is small, then the model produces, on average, a good set of predictions. However, even when $\overline{\text{MRE}}$ is small, there may be one or more predictions that can be very bad. Most researchers consider $\overline{\text{MRE}} \leq 0.20$ as acceptable for effort prediction models.

2.3 Prediction at Level r [PRED(r)]

Let k be the number of projects in a set of n projects whose MRE $\leq r$. Then we define this measure as

$$\text{PRED}(r) = k/n \tag{2.5}$$

For example, if PRED(0.25) = 0.83, then 83% of the predicted values fall within 25% of their actual values. Most researchers have concluded that an acceptable criterion for an effort prediction model is PRED(0.25) $\geq$ 0.75. Of course, this measure would also permit some extremely poor predicted values. In particular, there is no limit on the MRE of the other estimates which exceed 25% of the actual values.

2.4 The Relative Root Mean Square Error ($\overline{\text{RMS}}$)

Given a set of n projects, we define

$$\bar{E} = \frac{1}{n}\sum_{i=1}^{n} E_i$$

and

$$\text{RMS} = \left[\frac{1}{n}\sum_{i=1}^{n} (E_i - \hat{E}_i)^2\right]^{1/2}$$

The relative root mean square error is defined by

$$\overline{\text{RMS}} = \text{RMS}/\bar{E} \tag{2.6}$$

We consider an $\overline{\text{RMS}} \leq 0.25$ as acceptable.

2.5 A Comparison of Measures

Unfortunately, the three primary measures [namely, $\overline{\text{MRE}}$, PRED(r), and $\overline{\text{RMS}}$] are often not in agreement for a set of projects in the sense that one or

TABLE II

A COMPARISON OF MEASURES OF GOODNESS

Data set	No. projects	$\overline{\text{MRE}}$	PRED(0.25)	$\overline{\text{RMS}}$
1	23	0.16	0.78	0.34[a]
2	28	0.19	0.75	0.51[a]
3	15	0.16	0.80	0.30[a]
4	19	0.11	1.00	0.13
5	17	0.14	1.00	0.20
6	33	0.27[a]	0.64[a]	0.46[a]
7	40	0.24[a]	0.73[a]	0.95[a]
8	12	0.28[a]	0.83	0.12

[a] Measure is "not acceptable."

more may be unacceptable while others are acceptable (based on criteria detailed in the three previous sections). One model that we have tested led to the measures shown in Table II. From these projects it appears that the $\overline{\text{RMS}}$ measure is more conservative than the $\overline{\text{MRE}}$ measure, although this is not so for set 8. Moreover, the more heterogeneous the projects in a data set, the worse the $\overline{\text{RMS}}$ measure is likely to be. This conclusion is based on the fact that the $\overline{\text{RMS}}$ measure is acceptable only on sets 4, 5, and 8, which are the only homogeneous sets in the table. On the other hand, there appears to be closer agreement between $\overline{\text{MRE}}$ and PRED(0.25). Indeed, if we term the simultaneous satisfaction of the two measures

$$\overline{\text{MRE}} \leq 0.20 \text{ and } \text{PRED}(0.25) \geq 0.75 \tag{2.7}$$

as constituting acceptable performance, then the model performance is acceptable on sets 1 – 5 but not acceptable on sets 6 – 8. On the whole, however, the $\overline{\text{MRE}}$ and PRED(r) measures appear to suggest overall acceptable performance of the model while the $\overline{\text{RMS}}$ measure leaves the performance in some doubt. In the remainder of this article we shall base decisions on acceptable performance on the criterion defined by Eq. (2.7).

3. Historical – Experiential Models

Most of the cost estimation methods in common use today undoubtedly fall into this category. In its crudest form, one or more local experts are asked to make judgments about the effort required, either for the total project or for modules into which the project has been divided. In doing so, the experts rely on their own experience with similar projects or modules, on intuition, and possibly on historically maintained information about completed projects.

If more than one expert is involved, then a simple or weighted average of their estimates is taken as a "best" starting estimate. This is, of course, a very subjective procedure which is highly dependent on the competence and objectivity of the estimators. Clearly such a procedure runs the risk of overlooking some especially difficult subtasks, which may be unique to the current project. On the other hand, an expert can incorporate into his estimate unique strengths or weaknesses of the local organization that would be difficult for a general-purpose estimator, which is likely to be based on average organizational characteristics.

The expert judgment and analogy methods described above can be applied at either the overall system level or at the system component level. These are commonly referred to as top-down estimating (overall system) or bottom-up estimating (system components). In top-down estimating, the focus is on the system level and has the advantage that the cost of system level functions, such as integration, documentation, and management, will more likely be taken into consideration. In bottom-up estimating, the system is broken down into modules and/or components. An estimate is then made for each component. The estimates are summed to obtain an overall estimate after allowing for proper component integration. Bottom-up estimating has the apparent advantage that components are generally examined at a more detailed level, implying that a better estimate should follow. On the other hand, bottom-up estimating may overlook the effort required for system integration and testing. While all effort estimation techniques can theoretically be applied either as top-down or bottom-up procedures, some lend themselves more naturally to one of these procedures.

Of the historically based models that have been described in the literature, the TRW Wolverton model (Wolverton, 1974) is probably the best known. It derives the system development cost from a software cost matrix. An example is shown in Table III. The elements of the matrix are costs which are

TABLE III

AN EXAMPLE OF THE COST MATRIX

Type	Level of difficulty					
	OE 1	OM 2	OH 3	NE 4	NM 5	NH 6
1 Control	21	27	30	33	40	49
2 I/O	17	24	27	28	35	43
3 Pre/postprocessor	16	23	26	28	34	42
4 Algorithm	15	20	22	25	30	35
5 Data management	24	31	35	37	46	57
6 Time critical	75	75	75	75	75	75