INTERNATIONAL BUSINESS SCHOOLS COMPUTING QUARTERLY

A Publication of the International Business Schools Computer Users Group

Volume 4, Number 1

Spring 1992

The strengths and limitations of algorithmic approaches to estimating and managing software costs

Hossein Saiedian
Mansour Zand
Darryl Barney
University of Nebraska, Omaha

Abstract

To estimate and manage software costs is to measure and correct software activities to ensure that economical goals are achieved. Estimation requires the measurement of performance against plans and resources. Management involves the creation and maintenance of an environment where the software developer can perform effectively while efficiently using resources. Of the software cost-estimating techniques, algorithmic models provide the most economical approach. This paper examines algorithmic models and discusses their strengths and limitations in trying to understand and control software project cost. Algorithmic models can provide valuable management information by determining cost drivers and attributes in a systematic manner.

1. Introduction

A software project manager is responsible for, among other things, controlling the cost of a project. He must be able to make reasonable estimates of how much a software system (or part of it) will cost. Algorithmic cost models are perhaps most valuable to managers in helping with quantitative option analysis.

The history of software cost estimation dates back to the late 1960's and early 1970's. Hardware costs started decreasing while software project costs began to soar. Corporation management started devoting personnel and assets to determine cost estimation models and software cost drivers.

Two extensive studies performed in the 1960's and 70's shifted the software economics field into high gear. In 1964, the United States Air Force evaluated 169 software projects, collecting statistical information on 104 attributes (Boehm 1984). In 1972, the Software Measurement Project conducted by IBM analyzed sixty completed software development projects. Twenty-eight high-level languages and 66 computers were represented in the well-structured information data base (Walston and Felix 1977). By the late 1970's, many software cost estimation models had been developed by several corporations, bringing significant advances in the state of the art. These include the Doty Model (Wolverton 1974), the RADC database model (Schneider 1978), the Putnam SLIM model (Warburton 1983), and the COCOMO model (Boehm 1984). Many other models soon disappeared due mainly to their inherent restrictions. Few models have stood the test of time.

2. Estimation techniques

Barry W. Boehm, in his book Software Economics (1981) and various other papers (Boehm and Papaccio 1988; Boehm 1984), described several techniques for estimating software costs. These techniques may be summarized as follows (Boehm and Papaccio 1988; Boehm 1984; Sommerville 1989):

1. Algorithmic Models: These methods provide one or more algorithms which produce a software cost estimate as a function of a number of variables which are considered to be the major cost drivers.

2. Expert Judgment: This method involves consulting one or more experts, determining cost, and mediating any conflicts.

3. Analogy: This method involves reasoning by analogy from one or more completed projects to relate their actual cost to an estimate of the cost of a similar new project.

4. Parkinson: A Parkinson principle ("work expands to fill

the available volume") is invoked to equate the cost estimate to the available resources.

5. Price-to-Win: Here, the cost estimate is equated to the price believed necessary to win the job (or the schedule believed necessary to be first in the market with a new product, etc.).

6. Top-Down: An overall cost estimate for the project is derived from global properties of the software product. The total cost is then split up among the various components.

7. Bottom-Up: Each component of the software job is separately estimated, and the results aggregated to produce an estimate for the overall job.

Algorithmic models have become the standard tool in estimating software cost because they use the least subjective approach. Mathematical formulas are developed which best represent completed projects. These formulas can be extensively cross-examined and validated. This paper will examine two popular models: the COCOMO model and the Putnam model. Differences, similarities, strengths, and weaknesses will be discussed along with other curious findings of past models.

3. Algorithmic models

The algorithmic models provide the most scientific approach to software cost (and scheduling). These methods provide one or more algorithms which produce a software cost estimation as a function of a number of variables which are considered to be the major cost drivers.

3.1 The COCOMO model

The best documented software costing model is the COnstructive COst MOdel (COCOMO). It has three levels of difficulty. The basic model develops a quick first guess as to the cost of a project. An intermediate model requires deeper understanding and determination of specific cost drivers. The third model expands on the intermediate model, requiring even greater depth of understanding and analysis of companies' historical projects data. Only the basic and intermediate models will be discussed in this paper.

Step 1: Nominal effort estimation

First, the project must be classified into one of three modes. Organic mode projects come from stable, familiar, forgiving, and relatively unconstrained environments. These are the projects with small development teams, little communications overhead, and non-complex I/O or algorithmic formulation.

Semi-detached mode projects are an intermediate stage between organic mode projects and embedded mode projects (explained below) and represent more ambitious, unfamiliar, unforgiving, less stable projects. These projects would be made up of less experienced team members or members unfamiliar with related systems.

Embedded mode projects represent critical systems with tightly embedded constraints on software and hardware. Thus requirements modifications to get around software problems are not practical and software validation/verifications are high. The diverse nature of these projects implies that project team members may not have a great deal of experience in the particular application which is being developed.

By studying historical data for a given corporation, best fit equations can be developed for each of the three project modes. Boehm determined that for TRW software projects the following equations corresponded to the three specific modes (Boehm 1984):

Project Mode	Nominal Effort	Schedule
Organic	$MM = 2.4(DSI)^{1.05}$	$TDEV = 2.5(MM)^{0.38}$
Semi-detached	$MM = 3.0(DSI)^{1.12}$	$TDEV = 2.5(MM)^{0.35}$
Embedded	$MM = 3.6(DSI)^{1.20}$	$TDEV = 2.5(MM)^{0.32}$

where
MM = number of man-months required to complete a project, based on 152 hours of work time per month;

KDSI = thousands of delivered source instructions, based on one instruction per line; and

TDEV = development time required in calendar months.

The number of personnel required to complete the project in the number of months is determined by:

Number Persons N = MM / TDEV

To show the basic COCOMO model, assume that an organic mode software project has an estimated 30,000 delivered source instructions. The number of manmonths required for this project can be calculated via the effort equation:

$$MM = 2.4(30)^{1.05} = 85 \text{ Man-Months}$$

while the time required to complete the project is computed as:

$TDVE = 2.5(85)^{0.38} = 14$ Months

and the number of personnel required to complete the project in the time scale is:

$N = MM/TDVE = 85/14 = 7$ People

Now consider an embedded software project which would consist of 150,000 delivered source instructions:

$MM = 3.6(150)^{1.20} = 1471$ Man-Months
$TDVE = 2.5(1471)^{0.32} = 26$ Months
$N = 1471/26 = 57$ People

Step 2: Determine effort multipliers

COCOMO has 15 cost driver attributes. These attributes contribute to a project's level of difficulty and account for variations in a project's cost. Each attribute has a rating scale which determines a set of effort multipliers. These effort multipliers modify the Nominal Effort Estimate into a project-specific equation. Once again, all estimates for the multiplier can be modified to fit a corporation's past project experiences.

Boehm provides the user with three charts with which to determine the complexity ratings versus the type of module, cost driver ratings, and effort multiplier ratings (Boehm and Papaccio 1988). It would be too lengthy to outline and explain all of the mentioned tables in this paper; however, it is important to note what Boehm determined to be effort multipliers.

The fifteen effort multipliers are based on four aspects of a computer project. They include:

I. Product Attributes

RELY = Required software reliability
DATA = Data base size
CPLX = Product complexity

II. Computer Attributes

TIME = Execution time constraint
STOR = Main storage constraint
VIRT = Virtual machine volatility
TURN = Computer turnaround time

III. Personnel Attributes

ACAP = Analyst capability
AEXP = Applications experience
PCAP = Programmer capability
VEXP = Virtual machine experience
LEXP = Programming language experience

IV. Project Attributes

MODP = Use of modern programming
TOOL = Use of software tools
SCED = Required development schedule

Step 3: Estimate development effort

Next, the user estimates the development effort for the project by multiplying the nominal development effort (MM) by the product of the effort multipliers of the 15 cost driver attributes. The resulting equation for the project cost estimation is:

Estimated Development Effort = MM * Effort Multipliers

Step 4: Estimate related factors

COCOMO allows the user to breakdown the overall development effort cost into separate life-cycle phases (requirement analysis, architectural design, etc.) and into various types of project activity (test planning, management, etc.). These techniques are beyond the scope of this paper. See Boehm and Papaccio 1988, Boehm 1984, and Sommerville 1989 for more detail. Boehm's own comment about COCOMO, however, should be repeated (1981, p.32):

Today, a software cost estimation model is doing well if it can estimate software development costs within 20% of actual costs, 70% of the time, and on its own turf (that is, within the class of projects to which it has been calibrated).... This is not as precise as we might like, but it is accurate enough to provide a good deal of help in software engineering economic analysis and decision making.

3.2 Putnam estimation model

The Putnam Model (Putnam 1978) assumes a specific distribution of effort over the life of a software development project. This model is used more by software managers because its primary use is to predict costs and delivery schedules. It has become a significant tool because it uses readily obtainable manpower data in a manner that is easily understood. Delays and cost overruns are easily visualized and their effects determined (Warburton 1983).

As does the COCOMO model, the Putnam model relies heavily on historical data. Completed projects provide data on which to base a calibration of the model. The model's first use is to generate likely cost and schedule estimates on future projects. The second use is for predicting cost and schedules while a project is progressing. The model provides realistic projections that can be compared with actual milestones.

The curve shows the number of people assigned to the project at any given time. Relatively few people are assigned during design and coding phases. Manpower reaches a peak during implementation and falls off during testing and validation when fewer people are required.

One assumption of the Putnam model is that all projects will take on similar characteristics and that the curve plotted is characterized by a Rayleigh distribution. The Rayleigh distribution is a mathematical model which tries to determine the total manpower of a project as a function of time. It appears to fit wide ranges of software projects whether small or large. The following formula relates the number of delivered lines of code to effort and development time:

$$L = C_k K^{1/3} t_d^{4/3}$$

where

C_k is a state-of-technology constant reflecting the throughput constraints that impede the progress of the programmers (typical value could be 2,000 for poor software development environment while a value of 8,000 might be used for a more improved environments);

K is person-year effort over the entire life cycle of a software (development and maintenance); and

t_d is the development time in years.

The above formula can be rearranged to arrive at expressions for person-year development effort (K) and development time in year(t):

$$K = \frac{L^3}{C_k^3 T_d^4} \quad , \quad t_d = \frac{L^{3/4}}{C_k^{3/4} k^{1/4}}$$

Putnam (1978) demonstrates that a relatively small extension in delivery date can bring projected substantial savings in human effort applied to the project.
As previously mentioned, the Rayleigh curve was based on a particular case history of program projects. It is very useful for determining future work for already progressing programs; however, it lacks capabilities to estimate work done before the detail design and code cycle. Newer models have successfully altered the Rayleigh Curve (Parr 1980), achieving results which not only are similar to those of the Rayleigh model but also account for work contributed to the project preceding its official start date.

Since the entire manpower curve is supposed to be Rayleigh-shaped, the curve can be subdivided into several subcycles. The first major subcycle is the detail design and code cycle, which also should be Rayleigh-shaped. The manpower peak usually corresponds to the time period when the first system prototype is delivered. This is supposed to occur after approximately 40 percent of the total budget has been used. This stage is followed by a series of modifications and upgrades which constitute the other 60 percent of the total project expenditures (Warburton 1983).

Each subcycle then represents milestones of the project with which predictions can be made. Overruns and additional manning requests can be evaluated and the total manpower curve can be recalculated to determine total cost and schedule. Values for the technology constant (like the COCOMO Model) can be calibrated to cost attributes of past projects or to estimates of attributes for current projects.

There is an automated costing system based on the Rayleigh-Nordon curve and the Putnam Estimation Model. This system is called SLIM and enables the software planner to

- calibrate the local software development environment by interpreting historical data (supplied by the planner);

- create an information model of software by extracting basic characteristics of software, personnel, environment, etc.; and

- perform software sizing.

4. The strengths/limitations of algorithmic models

4.1 Strengths

Of all the cost-estimating techniques available, only Algorithmic Models use scientific reasoning. This allows the user to analyze, verify, and modify the formulas to best fit individual projects. Even though the other techniques use historical data, none allows for in-depth analysis of cost attributes or drivers.

Algorithmic Models have proven through extensive validation to be accurate. Using a sample of 63 projects, Boehm verified that the COCOMO model provides estimates which come within 20 percent of the actual cost of a project about 68 percent of the time (Boehm 1984). A further study was done with a sample of 46 projects by the Software Development Planning Group, Fujitsu Limited. After the deletion of two extreme cases, that study also verified that, with minor modifications, the COCOMO model estimated within 20 percent of the actual cost of a project about 68 percent of the time (Miyazaki and Kuniaki 1985).

A particular model's strength corresponds directly to the amount of modifications which the model allows. The COCOMO model is probably the most documented and studied model developed. It allows for tailoring of a wide variety of cost attributes. Earlier cost models which were less flexible proved to be less accurate. More recent cost-estimation models have been commercially developed which consider more cost attributes than the COCOMO model.

A simulated project problem was proposed by Rubin (1985) to compare four estimation models. The problem statement identified more than 28 areas of cost considerations. Of the models compared, none took into consideration all the cost drivers. The GECOMO model (COCOMO automated by GEC Software) only used 16 attributes. Newer improved models, like Rubin's own ESTIMACS, considered 24 of the 28 cost attributes. Improved outputs were also available. Models can provide results like risk analysis, financial break-even points, sensitivity, expected errors, maintenance costs, and best bid amounts.

4.2 Weaknesses

There are many critics of all the techniques for cost estimation. Comparing calculations from several algorithmic models usually results in large variations of manpower effort and project cost (Schneider 1978; Rubin 1985). Several fundamental stumbling blocks can cause any estimate to deviate from the actual cost. Several will be discussed.

Since algorithmic models are based on historical project data, they are only as good as the data collected. Faulty documentation could incorporate faulty data. Documentation is probably the worst task a project manager/programmer has to face. A manager not only has to manage the project but also has to allow his peers to inspect and criticize the results. Fujitsu Limited noted that one of the hardest problems of their research project was getting all the project managers to complete the documentation on a standardized questionnaire (Miyazaki and Mori 1985).

Errors with historical data can arise from the differences in viewpoints of the recorder and the information user. Standardized recording procedures must be installed to insure that all documents are compatible. Changes sometimes have to be made to the basic data elements to make sure historical data are compatible. Fujitsu Limited, for example, had to convert the Japanese man-month of 170 hours to the American man-month of 152 hours. Standardized definitions of terms also have to be determined, so that project classifications can be correctly determined. With the COCOMO model, changes in project modes or increased cost attributes or drivers can cause the estimate to be in error.

The fundamental problem of historical data is that since it is based on past projects, not future or current projects, it does not allow for advances in software tools, languages, hardware, or software engineering. The effect of these improvements cannot be accurately calculated until future projects have been completed and their data documented. Examples of how dramatically new technology can decrease software cost are evident in the differences of project cost before and after the advent of high level languages (McGowen 1980; Wasserman 1988). Future automated software development tools could also disqualify past estimates.

Historical data is hard to obtain since numerous projects must first be completed from which to amass enough data. Corporations that are just starting have no real data to use. Historical data is also software-team dependent. Results from one software team may not develop good estimates for another team.

Another fundamental flaw with algorithmic models is that they rely heavily on some estimated metric of the finished software project. The most widely used metric is the total lines of code of the finished project. Since cost-estimation is most critical early in the software development, determining the amount of code for a project breaking new ground can be very difficult (Sommerville 1989).

4.3 Observations

In the COCOMO Model, nominal effort for a program is determined by using best fit equations with the variable being thousands of delivered source instructions. The Putnam SLIM model used number of people versus time. To accurately determine the number of people required for a project, managers must determine the size of the project at an early stage of development. All models are directly dependent on the amount of code required.

What describes a line of source code also influences the cost-estimation. Boehm describes a line of code as any line of source text, irrespective of the actual number of instructions on that line (Boehm 1984). Therefore, if there is more than one source instruction on a line, it is considered one line. Adversely, if one source instruction is spread over five lines, then it would be considered five instructions. Comments and undelivered support software are excluded from the lines of code instructions, even though the effort involved in these may consume many man hours.

Project complexity can also affect the final amount of lines of source instruction. Program length is a poor metric for determining complexity. Large programs may have straightforward mathematical algorithms, while a short program may have a complex algorithm intertwined with complex data transactions. Complexity-estimating models have been developed; but as with estimating lines of code, it is difficult and extremely critical to estimate complexity during the early stages of the project development.

A change in requirements by the customer can also have a dramatic effect on project cost. If the changes are requested early in the development or do not require major modifications, cost estimates may not be affected. Otherwise, major rewrites would increase cost estimates, though they would not be reflected in the total lines of source instructions delivered.

New questions have arisen with the advent of the usage of reusable code. Such pre-developed and tested software code which can dramatically reduce nominal effort (Boydston 1984).

Cost estimates can never determine all the cost attributes or drivers that may affect program projects. Technological or economic changes are at best difficult to predict. Thus, algorithmic models can never be totally perfect based on the dynamic environment of software.

Though they have limitations, Algorithmic Models can be useful tools in determining costs and cost drivers of software projects. For example, the COCOMO Model allows for modifications of cost attributes. Therefore, it is possible for management to manipulate attributes to determine maximum cost-effective approaches. Corporations can determine what minimal investment in tools, hardware, training, etc., will generate the greatest returns. Models are tools for determining, understanding, and controlling software costs (Boehm and Papaccio 1988; Boehm 1984; Walston and Felix 1977; Warburton 1983; Daly 1977; Duncan 1988).

5. Conclusions

Algorithmic models can be used to demonstrate software engineering principles. The Putnam SLIM Model provides a means of understanding the myth that throwing more people on a late project only makes it later. By increasing the number of people in the people versus time formula, the project development curve will be increased. The cycle peak will be delayed which in turn delays the project completion date and increases cost (Warburton 1983).

The user must have a good working understanding of the limitations of the model since faulty assumptions may lead to erroneous conclusions. It has been widely suggested that several different estimating techniques be used. Differences in their results should be checked and reanalyzed until the differences are resolved or at least understood (Sommerville 1989).

Finally, Algorithmic Models are only as good as the historical data upon which they are based. The available information for research is limited. For greater understanding and controlling of software development cost, corporations and government agencies must publish their software development techniques and project records. This would provide the large database of projects needed for the development of more precise algorithmic models.

References

[1] Boehm, B., and P. Papaccio. "Understanding and Controlling Software Costs," IEEE Transactions on Software Engineering, Vol. 14, 10, (Oct. 1988), pp. 1462-1477.
[2] Boehm, B. Software Engineering Economics, Prentice-Hall, 1981.

[3] Boehm, B. "Software Engineering Economics," IEEE Transactions on Software Engineering, Vol.SE-10, No. 1 (Jan. 1984), pp. 4-21.

[4] Boydston, R.E. "Programming Cost Estimate: Is It Reasonable?," Proceedings, Eighth International Conference on Software Engineering, (1984), pp. 153-159.

[5] Daly, F.B. "Management of Software Development," IEEE Transactions Software Engineering, Vol. 3, (1977), pp. 230-242.

[6] Duncan, A.S. "Software Development Productivity, Tools and Metrics," Proceedings, Twelfth International Conference on Software Engineering, (1988), pp. 41-48.

[7] McGowan, M.J. "High-level Microcomputer Languages Slash Software Development Cost," Control Engineering, Vol. 27, 4, (Apr. 1980), pp. 53-58.

[8] Miyazaki, Y. and Kuniaki Mori, "COCOMO Evaluation and Tailoring," Proceedings, Ninth International Conference on Software Engineering, (1985), pp. 292-299.

[9] Parr, F.N. "An Alternative to Rayleigh Curve Model for Software Development Effort," IEEE Transactions on Software Engineering, Vol. 6, 3, (1980), pp. 291-296.

[10] Putnam, L. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," IEEE Transactions on Software Engineering, Vol. 4, No. 4, (1978), pp. 345-361.

[11] Rubin, H.A. "A Comparison of Cost Estimation Tools," Proceedings, Ninth International Conference on Software Engineering, (1985), pp. 174-180.

[12] Schneider, V. "Predictions of Software Effort and Project Duration - Four New Formulas," SIGPLAN Notices , Vol. 13, 6, (Jun. 1978), pp. 49-59.

[13] Sommerville, Ian. Software Engineering. 3rd Ed., Addison-Wesley Publishing Company, (1989), pp. 515-533.

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

[15] Warburton, R.D.H. "Managing and Predicting The Costs of Real-time Software," IEEE Transactions on Software Engineering. Vol. 9, 5, (Sep. 1983), pp. 562-568.

[16] Wasserman, A.L. "Implications of Hardware Advances for Software Development," Proceedings. Twelfth International Conference on Software Engineering. (1988), p. 250.

[17] Wolverton, R.W. "The Cost of Developing Large Scale Software," IEEE Transactions Software Engineering, Vol. 23, 6, (1974).