Literature Survey on Algorithmic Methods for Software Development Cost Estimation

Mrs. Shubhangi Mahesh Potdar¹
Assistant professor, IBMRD,
Ahmednagar, India
Email:shubhangipotdar@rediffmail.com

Dr. Manimala Puri²
Director, JSPM Group
of Institutes,
Pune, India
Email:manimalap@yahoo.com

Mr. Mahesh P. Potdar³
Associate professor, IMSCD & R,
Ahmednagar, India
Email:maheshpotdar@rediffmail.com

Abstract

Cost estimation is the key part of projects management. Accurate cost estimation plays main role to complete the project within time and budget. There are many software cost estimation methods, which can be classified into two types’ algorithmic methods and non algorithmic methods. This classification is depends upon use of mathematical expressions for estimating cost. In Algorithmic Cost Estimation there is use of mathematical expressions for estimating cost associated with software development. While in Non Algorithmic Cost Estimation there is no use of mathematical expressions for software development cost estimation

Inaccurate cost estimation may lead to project failure, huge overruns and performance compromises as a consequence.

Considerable studies are now directed at constructing, evaluating and selecting better software cost estimation models and tools for specific software development projects. This paper gives an overview of Algorithmic Cost Estimation models and then discusses their advantages and disadvantages. No one method is necessarily better or worse than the other, in fact, their strengths and weaknesses are often complimentary to each other. Finally, the recommendations for selecting appropriate cost estimation models are given and a combination method is recommended.

Keywords: Algorithmic Models, COCOMO Models, Function Point Analysis Based Methods, Non-Algorithmic, Putnam model, SLOC, KDSI.

1. Introduction

Software Cost estimation is generally used by the system analyst to estimate needed resources and required schedules for software development projects. There are various parameters of cost estimation like size, time, effort etc. The software estimation process includes estimating the size of the software product to be developed, estimating the effort required, developing preliminary project schedules, and finally, estimating overall cost of the project.

Software Cost estimation is very difficult task in software development because each software to be developed is different from others.

Accuracy of software cost estimation depends on the nature of the estimating methodologies. Software cost estimation is a continuing activity, which starts at the proposal stage and continues through the lift time of a project. Continual cost estimation is to ensure that project is in line with the budget.

Software cost estimation methods can be classified into two types, algorithmic methods and non algorithmic methods. This paper gives an overview of Algorithmic Cost Estimation models and then discusses their advantages and disadvantages. To understand their strengths and weaknesses is very important when you want to estimate your projects.

Why does IT Project Fails [11]:
1. Poor User Input
2. Stakeholder Conflicts
3. Vague Requirements
4. Poor Cost and Schedule Estimation
5. Skills that Do Not Match the Job
6. Hidden Costs of Going "Lean and Mean"
7. Failure to Plan
8. Communication Breakdowns
9. Poor Architecture
10. Late Failure Warning Signals

Ten Reasons Software Projects Succeed [12]:
1. High level sponsorship
2. End user participation
3. Focus
4. Developer has understanding of the client's business
5. Client has understanding of the development process
6. Project management
7. Technical competence and maturity of the developer
8. Shared reality in a flexible environment
9. Open communications
10. Prompt payment

This paper begins with the introduction. Section II covers the algorithmic methods for cost estimation. Section III covers recommendation for selecting algorithmic methods. Section IV is the conclusion. Section V covers references.

2. ALGORITHMIC METHODS FOR COST ESTIMATION

The algorithmic method is designed to provide some mathematical equations to perform software estimation. These mathematical equations are based on research and historical data and use inputs such as Source Lines of Code (SLOC), number of functions to perform, and other cost drivers such as language, design methodology, skill-levels, risk assessments, etc. The algorithmic methods have been largely studied and there are a lot of models have been developed, such as COCOMO models, Putnam model, and Function points based models.

General advantages of algorithmic methods:
1. It is able to generate repeatable estimations.
2. It is easy to modify input data, refine and customize formulas.
3. It is efficient and able to support a family of estimations or a sensitivity analysis.
4. It is objectively calibrated to previous experience.

General disadvantages of algorithmic methods:
1. It is unable to deal with exceptional conditions, such as exceptional personnel in any software cost estimating exercises, exceptional teamwork, and an exceptional match between skill-levels and tasks.
2. Poor sizing inputs and inaccurate cost driver rating will result in inaccurate estimation.

2.1. COCOMO Models:

The Constructive Cost Model (COCOMO) was developed by Barry Boehm of TRW and published in 1981. Based on his analysis of 63 software-development projects; Boehm developed an easy-to-understand model that predicts the effort and duration of a project, based on inputs relating to the size of the resulting systems and a number of “cost drivers” that Boehm believes affect productivity [10]. It is an algorithmic approach to estimate the cost of a software project. By using COCOMO you can calculate the amount of effort and the time schedule for projects. From these calculations you can then find out how much staffing is required to complete a project on time. COCOMO's main metric used for calculating these values is lines of code (denoted KLOC for COCOMO II, or KDSI for COCOMO 81 and measured in thousands), function points (FP), or object points (OP).

Boehm proposed three levels of the model: basic, intermediate, detailed.

- The basic COCOMO model is a single-valued, static model that computes software development effort (and cost) as a function of program size expressed in estimated thousand delivered source instructions (KDSI).
- The intermediate COCOMO model computes software development effort as a function of program size and a set of fifteen "cost drivers" that include subjective assessments of product, hardware, personnel, and project attributes.
- The advanced or detailed COCOMO model incorporates all characteristics of the intermediate version with an assessment of the cost driver’s impact on each step (analysis, design, etc.) of the software engineering process.

One very widely used algorithmic software cost model is the Constructive Cost Model (COCOMO). The basic COCOMO model has a very simple form:

\[ \text{MAN-MONTHS} = K1 \times (\text{Thousands of Delivered Source Instructions})^{K2} \]

Where, K1 and K2 are two parameters dependent on the application and development environment. [4]

The first version of COCOMO model was originally developed in 1981. Now, it has been experiencing increasing difficulties in estimating the cost of software developed to new life cycle processes and capabilities including rapid-development process model, reuse-driven approaches, object-oriented approaches and software process maturity initiative[10].
The Second version, COCOMO 2.0 are a tailor able family of software size models, involving object points, function points and source lines of code; nonlinear models for software reuse and reengineering; an exponent-driver approach for modeling relative software diseconomies of scale; and several additions, deletions, and updates to previous COCOMO effort-multiplier cost drivers. This new model is also serving as a framework for an extensive current data collection and analysis effort to further refine and calibrate the model's estimation capabilities.

One of Boehm’s original motivation for creating COCOMO was to decrease the number of errors managers make when estimation software projects. At the early software development stage such as investigation and inception phase, the characteristics of software system are unknown; the nature of the processes, team, and personnel experiences are still unclear; the degree of understanding architecture, requirements, and constraints are low. When the software project goes into further development phase, more knowledge of the project is available so predictions better approximate the actual cost.

COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. The first level, Basic COCOMO is good for quick, early, rough order of magnitude estimates of software costs, but its accuracy is limited due to its lack of factors to account for difference in project attributes (Cost Drivers).

Intermediate COCOMO takes these Cost Drivers into account and Detailed COCOMO additionally accounts for the influence of individual project phases. Basic COCOMO computes software development effort (and cost) as a function of program size. Program size is expressed in estimated thousands of lines of code (KLOC).

COCOMO applies to three classes of software projects: Organic projects - "small" teams with "good" experience working with "less than rigid" requirements Semi-detached projects - "medium" teams with mixed experience working with a mix of rigid and less than rigid requirements.

Embedded projects - developed within a set of "tight" constraints (hardware, software, operational etc.)

Advantages:
1. COCOMO II is an industry standard
2. Very profound information is easy available
3. Clear and effective calibration process by combining Delphi with algorithmic cost estimation techniques
4. Backwards compatibility
5. Various extensions for almost every purpose are available
6. Tool support (also for the various extensions)

Drawbacks:
1. The 'heart' of COCOMO 2.0 is still based on a waterfall process model
2. Most of the extensions are still experimental and not fully calibrated till now
3. Duration calculation for small projects is unreasonable [15]

2.2. Putnam Model:
Another popular software cost model is the Putnam model. The form of this model is:

Technical constant C= size * B^{1/3} * T^{4/3}

Total Person Months B=1/T^4 * (size/C)^3

T= Required Development Time in years

Size is estimated in LOC

Where, C is a parameter dependent on the development environment and it is determined on the basis of historical data of the past projects.

Rating: C=2,000 (poor), C=8000 (good) C=12,000 (excellent) [3].

The Putnam model is very sensitive to the development time, decreasing the development time can greatly increase the person-months needed for development.

An estimated software size at project completion and organizational process productivity is used. Plotting effort as a function of time yields the Time-Effort Curve. The points along the curve represent the estimated total effort to complete the project at some time. One of the distinguishing features of the Putnam model is that total effort decreases as the time to complete the project is extended. This is normally represented in other parametric models with a schedule relaxation parameter.

This estimating method is fairly sensitive to uncertainty in both size and process productivity.

Advantages:
One of the key advantages to this model is the simplicity with which it is calibrated. Most software organizations, regardless of maturity level can easily collect size, effort and duration (time) for past projects. Process Productivity, being exponential in nature is typically converted to a linear productivity index an organization can use to track their own changes in productivity and apply in future effort estimates.

Disadvantages:

One significant problem with the PUTNAM model is that it is based on knowing, or being able to estimate accurately, the size (in lines of code) of the software to be developed. There is often great uncertainty in the software size. It may result in the inaccuracy of cost estimation.

Time is very dominating factor in Putnam model. Putnam model is basically based on only two variables which are time and size. It is not considering all other aspects of software development life cycle. Whereas in COCOMO II we are getting more nearer results because it is considering almost all aspects of SDLC.

2.3. Function Point Analysis Based Methods:

The Function Point Analysis is another method of quantifying the size and complexity of a software system in terms of the functions that the system delivers to the user. A number of proprietary models for cost estimation have adopted a function point type of approach, such as ESTIMACS and SPQR/20.

The function point measurement method was developed by Allan Albrecht at IBM and published in 1979. He believes function points offer several significant advantages over SLOC counts of size measurement. There are two steps in counting function points:

- Counting the user functions. The raw function counts are arrived at by considering a linear combination of five basic software components: external inputs, external outputs, external inquiries, logic internal files, and external interfaces, each at one of three complexity levels: simple, average or complex. The sum of these numbers, weighted according to the complexity level, is the number of function counts (FC).
- Adjusting for environmental processing complexity. The final function points is arrived at by multiplying FC by an adjustment factor that is determined by considering 14 aspects of processing complexity. This adjustment factor allows the FC to be modified by at most 35% or ±35%. [4]

The collection of function point data has two primary motivations. One is the desire by managers to monitor levels of productivity. Another use of it is in the estimation of software development cost.

There are some cost estimation methods which are based on a function point type of measurement, such as ESTIMACS and SPQR/20. SPQR/20 is based on a modified function point method. Whereas traditional function point analysis is based on evaluating 14 factors, SPQR/20 separates complexity into three categories: complexity of algorithms, complexity of code, and complexity of data structures. ESTIMACS is a propriety system designed to give development cost estimate at the conception stage of a project and it contains a module which estimates function point as a primary input for estimating cost. [1]

Advantages:

1. Standards are established and reviewed frequently.
2. Resulting metrics are logical and straightforward.
3. Counting resources are available from requirements stage and applicable for full life-cycle analysis.
4. Technology, platform and language independent.
5. Objectively defines software application from the user’s perspective.
6. Function points can be estimated from requirements specifications or design specifications, thus making it possible to estimate development cost in the early phases of development.
7. Non-technical users have a better understanding of what function points are measuring since function points are based on the system user's external view of the system [4].

Disadvantages:

1. Largely a manual process.
2. Accurate counting requires in-depth knowledge of standards.
3. Some variations exist that are not standardized (Mark II, 3D, full, feature points, object points, etc.).
4. Not as much historical data available as SLOC.

Sometimes backfiring, derived from SLOC can be inaccurate and misleading.
2.4. The selection of Estimation methods:

From the above comparison, we know no one method is necessarily better or worse than the other, in fact, their strengths and weaknesses are often complimentary to each other. According to the experience, it is recommended that a combination of models is useful to get reliable, accurate cost estimation for software development. [1]

For large, lesser known projects, it is better to use algorithmic model. In this case, many researchers recommend the estimation models that do not required SLOC as an input. I think COCOMO2.0 is the first candidate because COCOMO2.0 model not only can use Source lines of code (SLOC) but also can use Object points, unadjusted function points as metrics for sizing a project. If we approach cost estimation by parts, we may use expert judgment for some known parts. This way we can take advantage of both: the rigor of models and the speed of expert judgment or analogy. Because the advantages and disadvantages of each technique are complementary, a combination will reduce the negative effect of any one technique, augment their individual strengths and help to cross-check one method against another.

2.5. Use of Cost Estimation Methods:

It is very common that we apply some cost estimation methods to estimate the cost of software development. But what we have to note is that it is very important to continually re-estimate cost and to compare targets against actual expenditure at each major milestone. This keeps the status of the project visible and helps to identify necessary corrections to budget and schedule as soon as they occur.

At every estimation and re-estimation point, iteration is an important tool to improve estimation quality. The estimator can use several estimation techniques and check whether their estimates converge. The other advantages are as following:

- Different estimation methods may use different data. This results in better coverage of the knowledge base for the estimation process. It can help to identify cost components that cannot be dealt with or were overlooked in one of the methods.
- Different viewpoints and biases can be taken into account and reconciled. A competitive contract bid, a high business priority to keep costs down, or a small market window with the resulting tight deadlines tends to have optimistic estimates. A production schedule established by the developers is usually more on the pessimistic side to avoid committing to a schedule and budget one cannot meet.

It is also very important to compare actual cost and time to the estimates even if only one or two techniques are used. It will also provide the necessary feedback to improve the estimation quality in the future. Generally, the historical data base for cost estimation should be set up for future use.

Identifying the goals of the estimation process is very important because it will influence the effort spent in estimating, its accuracy, and the models used. Tight schedules with high risks require more accurate estimates than loosely defined projects with a relatively open-ended schedule. The estimators should look at the quality of the data upon which estimates are based and at the various objectives. [1]

2.6. Strength and Weakness of the Existing Cost Estimation Methods:

Here, we describe the advantages and disadvantages of existing cost estimation methods. This description could be useful for choosing an appropriate method in a particular project. Table I. shows a comparison of mentioned methods for estimation. For doing comparison, the popular existing estimation methods have been selected.

Table I. Advantages and Disadvantages of existing methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>COCOMO</td>
<td>Algorithmic</td>
<td>Clear results, very common</td>
<td>Much data is required. It is not suitable for any project.</td>
</tr>
<tr>
<td>Putnam Model</td>
<td>Algorithmic</td>
<td>Putnam model is basically based on only two variables which is time and size</td>
<td>It is not considering all other aspects of software development life cycle</td>
</tr>
<tr>
<td>Function Point Analysis</td>
<td>Algorithmic</td>
<td>Language free, Its results are better than SLOC</td>
<td>Mechanization is hard to do, quality of output are not considered</td>
</tr>
</tbody>
</table>

3. Some recommendations:

1. Do not depend on a single cost or schedule estimate.
2. Use several estimating techniques or cost models, compare the results, and determine the reasons for any large variations.
3. Document the assumptions made when making the estimates.
4. Monitor the project to detect when assumptions that turn out to be wrong jeopardize the accuracy of the estimate.
5. Improve software process: An effective software process can be used to increase accuracy in cost estimation in a number of ways.

6. Maintaining a historical database[1]

4. Conclusions

The accurate prediction of software development costs is a critical issue to make the good management decisions and accurately determining how much effort and time a project required for project managers as well as system analysts and developers. There are many software cost estimation methods available including algorithmic methods, non-algorithmic methods. No one method is necessarily better or worse than the other, in fact, their strengths and weaknesses are often complimentary to each other. To understand their strengths and weaknesses is very important when you want to estimate your projects.

5. References:


