International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 7, September - 2012

Software Cost Models

Y.Sangeetha M.Tech (Ph.d)
Asst.Professor,
VRSEC,
Vijayawada.

P.Madhavi Latha
Asst.Professor
VRSEC
Vijayawada.

Dr R.Satya Prasad
Associate Professor
Acarya Nagarjuna University
Vijayawada.

Abstract

Software cost estimation is the process of predicting the effort required to develop a software system. Large numbers of estimation models have been proposed over the last 30 years. This paper provides a comparison of existing software cost estimation methods including the recent advances in the field. I have highlighted the cost estimation models that have been proposed and used successfully. Models may be classified into 2 major categories: algorithmic and non-algorithmic. Each has its own strengths and weaknesses. This paper compares the most popular algorithmic models used to estimate software costs [SLIM, COCOMO, Function Points, SEER-SEM and so on). A key factor in selecting a cost estimation model is the accuracy of its estimates.

I. INTRODUCTION

A cost model is a set of mathematical relationships arranged in a systematic sequence to develop a cost methodology in which outputs, namely cost estimates, are derived from inputs. These inputs include quantities and prices. Cost models can vary from a simple one-formula model to an extremely complex model that involves hundreds or even thousands of calculations.
Cost models can be classified in several ways. One basis for classification would be the complexity of manipulation of the inputs, secondly according to the function they serve and lastly according to the likelihood of repetitive use. Earlier various software cost estimation models have been suggested and studied by many researchers (Kim and Lee, Kafura and Henry, Kaur, Mittal and Parkash, Maxwell, Brian and Smith).This paper describes the comparative analysis of all the existing cost models. In this study, we investigate the estimation accuracies of each model.

II. RELATED WORK

Various Cost models have been examined e.g. COCOMO and FPA and I review the independent work done by various researchers who have investigated these models. My research is concerned with the comparative study of all existing models using actual project data. An important task in software project management is to understand and control critical variables that influence the software effort [5]. Some recent study is also done in the field of ―"Analogy based Estimations". Analogy based estimations compare the similarities between the projects whose effort is to be estimated with all the historical projects. In other words, it tries to identify that historical project which is most similar to the project being estimated [6].

III. NEED FOR COST ESTIMATION MODELS

Cost estimation is one of the most challenging tasks in project management. It is to accurately estimate needed resources and required schedules for software development projects. The software estimation process includes estimating the size of the software product to be produced, estimating the effort required, developing preliminary project schedules, and finally, estimating overall cost of the project. Accurate cost estimation is important because:

- It can help to classify and prioritize development projects with respect to an overall business plan.
- It can be used to determine what resources to commit to the project and how well these resources will be used.
- It can be used to assess the impact of changes and support replanning.
- Projects can be easier to manage and control when resources are better matched to real needs.
- Customers expect actual development costs to be in line with estimated costs.

IV. TYPES OF COST MODELS

COCOMO MODEL

Basic COCOMO Model

The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions:

$$\textbf{Effort} = a_1 \times (\text{KLOC})^{a_2}\ \text{PM}$$

$$T_{dev} = b_1 \times (\text{Effort})^{b_2}\ \text{Months}$$

Where, • KLOC is the estimated size of the software product expressed in Kilo Lines of Code,

- a_1, a_2, b_1, b_2 are constants for each category of software products,
- Tdev is the estimated time to develop the software, expressed in months,
- Effort is the total effort required to develop the software product, expressed in person months (PMs).

Estimation of development effort

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

Organic : Effort = $2.4(KLOC)^{1.05}$ PM

Semi-detached : Effort = $3.0(KLOC)^{1.12}$ PM

Embedded : Effort = $3.6(KLOC)^{1.20}$ PM

Estimation of development time

For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

Organic : $T_{dev} = 2.5(Effort)^{0.38}$ Months

Semi-detached: $T_{dev} = 2.5(Effort)^{0.35}$ Months

Embedded : $T_{dev} = 2.5(Effort)^{0.32}$ Months

Drawbacks of Basic COCOMO:

- It can be observed that the development time is a sublinear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately.
- Accurate Estimation is not obtained as a host of other project parameters besides the product size affect the effort required to develop the product as well as the development time.

Intermediate COCOMO model

The basic COCOMO model assumes that effort and development time are functions of the product size alone. However, a host of other project parameters besides the product size affect the effort required to develop the product as well as the development time. Therefore, in order to obtain an accurate estimation of the effort and project duration, the effect of all relevant parameters must be taken into account. The intermediate COCOMO model recognizes this fact and refines the initial estimate obtained using the basic COCOMO expressions by using a set of 15 cost drivers (multipliers) based on various attributes of software development. Boehm requires the project manager to rate these 15 different parameters for a particular project on a scale of one to three. Then, depending on these ratings, he suggests appropriate cost driver values which should be multiplied with the initial estimate obtained using the basic COCOMO. In general, the cost drivers can be classified as being attributes of the following items:

Product: The characteristics of the product that are considered include the inherent complexity of the product, reliability requirements of the product, etc.
Computer: Characteristics of the computer that are considered include the execution speed required, storage space required etc.
Personnel: The attributes of development personnel that are considered include the experience level of personnel, programming capability, analysis capability, etc.
Development Environment: Development environment attributes capture the development facilities available to the developers. An important parameter that is considered is the sophistication of the automation (CASE) tools used for software development.

Drawbacks of Intermediate COCOMO:

- Consider a software product as a single homogeneous entity. However, most large systems are made up several smaller sub-systems.

Complete COCOMO model

A major shortcoming of both the basic and intermediate COCOMO models is that they consider a software product as a single homogeneous entity. However, most large systems are made up several smaller sub-systems. These sub-systems may have widely different characteristics. For example, some sub-systems may be considered as organic type, some semidetached, and some embedded. Not only that the inherent development complexity of the subsystems may be different, but also for some subsystems the reliability requirements may be high, for some the development team might have no previous experience of similar development, and so on. The complete COCOMO model considers these differences in characteristics of the subsystems and estimates the effort and development time as the sum of the estimates for the individual subsystems. The cost of each subsystem is estimated separately. This approach reduces the margin of error in the final estimate.
The following development project can be considered as an example application of the complete COCOMO model. A distributed Management Information System (MIS) product for an organization having offices at several places across the country can have the following sub-components:
• Database part
• Graphical User Interface (GUI) part
• Communication part

Of these, the communication part can be considered as embedded software. The database part could be semi-detached software, and the GUI part organic software. The costs for these three components can be estimated separately, and summed up to give the overall cost of the system.

SEER-SEM MODEL

The System Evaluation and Estimation of Resources – Software Estimating Model (SEER-SEM) began with the Jensen model and diverged significantly in the early 1990s. SEER-SEM is composed of a group of models working together to provide estimates of effort, duration, staffing, and defects. Supported sizing metrics include source lines of code (SLOC), function-based sizing (FBS) and a range of other measures. They are translated for internal use into effective size (S_e). S_e is a form of common currency within the model and enables new, reused, and even commercial off-the-shelf code to be mixed for an integrated analysis of the software development process. The generic calculation for Se is:

$$S_e = NewSize + ExistingSize \times (0.4 \times Redesign + 0.25 \times Reimpl + 0.35 \times Retest)$$

In SEER-SEM, all size metrics are translated to S_e, including those entered using Function Based Sizing (FBS). After FBS is translated into function points, it is then converted into S_e as:

$$S_e = Lx * (AdjFactor * UFP)^{\frac{Entropy}{1.2}}$$

where, *Lx* is a language-dependent expansion factor. *AdjFactor* is the outcome of calculations involving other factors mentioned above. *Entropy* ranges from 1.04 to 1.2 depending on the type of software being developed.
The basic effort equation is:

$$K = D^{0.4} (S_e/C_{te})^{1.2}$$

where, *Se* is effective size – introduced earlier. C_{te} is effective technology – a composite metric that captures factors relating to the efficiency or productivity with which development can be carried out [1].

Advantages of SEER-SEM:

- Allows probability level of estimates, staffing and schedule constraints to be input as independent variables.
- Facilitates extensive sensitivity and trade-off analyses on model input parameters.
- Organizes project elements into work breakdown structures for convenient planning and control.
- Displays project cost drivers.
- Allows the interactive scheduling of project elements on Gantt charts.
- Builds estimates upon a sizable knowledge base of existing projects

Drawbacks of SEER-SEM:

- There are over 50 input parameters related to the various factors of a project, which increases the complexity of SEER-SEM, especially for managing the uncertainty from these outputs.
- The specific details of SEER-SEM increase the difficulty of discovering the nonlinear relationship between the parameter inputs and the corresponding outputs. Overall, these two major limitations can lead to a lower accuracy in effort estimation by SEER-SEM [4].

PRICE-S

The PRICE-S model was originally developed at RCA for use internally on software projects such as some that were part of the Apollo moon program. It was then released in 1977 as a proprietary model and used for estimating several US DoD, NASA and other government software projects. The model equations were not released in the public domain, although a few of the model's central algorithms were published in [Park 1988]. The tool continued to become popular and is now marketed by PRICE Systems, which is a privately held company formerly affiliated with Lockheed Martin. As published on PRICE Systems website (http://www.pricesystems.com), the PRICE-S Model consists of three submodels that enable estimating costs and schedules for the development and support of computer systems. These three submodels and their functionalities are outlined below:

The Acquisition Submodel: This submodel forecasts software costs and schedules. The model covers all types of software development, including business systems, communications, command and control, avionics, and space systems. PRICE-S addresses current software issues such as reengineering, code generation, spiral development, rapid development, rapid prototyping, object-oriented development, and software productivity measurement.

The Sizing Submodel: This submodel facilitates estimating the size of the software to be developed. Sizing can be in SLOC, Function Points and/or Predictive Object Points (POPs). POPs is a new way of sizing object oriented development projects and was introduced in [Minkiewicz 1998] based on previous work one in Object Oriented (OO) metrics done by Chidamber et al. and others [Chidamber and Kemerer 1994; Henderson-Sellers 1996].

The Life-cycle Cost Submodel: This submodel is used for rapid and early costing of the maintenance and support phase for the software. It is used in conjunction with the Acquisition Submodel, which provides the development costs and design parameters.

PRICE Systems continues to update their model to meet new challenges. Recently, they have added Foresight 2.0, the newest version of their software solution for forecasting time, effort and costs for commercial and non-military government software projects.[8]

SLIM

The SLIM model developed by Putnam is based on a Norden/Rayleigh manpower distribution and his finding in analyzing many completed projects [Putnam and Myers 1992].The central part

$$S = C_k * Effort^{1/3} * t_d^{4/3}$$

Where, *Effort* is in person-months, t_d is the software delivery time; C_k is a productivity environment factor.

A Manpower Buildup Index (MBI) and a Technology Constant or Productivity factor (PF) are used to influence the shape of the curve. SLIM can record and analyze data from previously completed projects which are then used to calibrate the model; or if data are not available then a set of questions can be answered to get values of MBI and PF from the existing database. The productivity environment factor reflects the development capability derived from historical data using the software equation. The size S is in LOC and the *Effort* is in person-years. Another relation found by Putnam is:

$$Effort = D_0 * t_d^3$$

Where, D_0 is the *manpower build-up parameter* which ranges from 8 (entirely new software with many interfaces) to 27 (rebuilt software).

Putnam's model is used in the SLIM software tool based on this model for cost estimation and manpower scheduling.

Advantages of SLIM:

- Uses linear programming to consider development constraints on both cost and effort.
- SLIM has fewer parameters needed to generate an estimate over COCOMO'81 and COCOMO'II

Drawbacks of SLIM:

- Estimates are extremely sensitive to the technology factor
- Not suitable for small projects

COPMO MODEL

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

$$\text{Effort} = Ep + Ec$$

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

$$E_p = a + b.KSLOC$$

where a and b are constant calculated with the best fit method. The second portion of the model $_{Ec}$ represents the cost of coordinating multiple programmers' effort. E_c is calculated as:

$$E_c = c.P^d$$

Where P is the average number of personnel, and c and d are constants calculated with the best fit method.
The COPMO model incorporates the simple intuition that the effort required to develop a system increases with the system's size. The COPMO model, in addition to the size component, explicitly accounts for the communication and management overhead associated with the large development staffs.

FUNCTION POINT ANALYSIS (FPA)

FPA begins with the decomposition of a project or application into its data and transactional functions. The data functions represent the functionality provided to the user by attending to their internal and external requirements in relation to the data, whereas the transactional functions describe the functionality provided to the user in relation to the processing this data by the application.[3]
The data functions are:
1. Internal Logical File (ILF)
2. External Interface File (EIF)

The transactional functions are:
1. External Input (EI)

2. External Output (EO)
3. External Inquiry (EI)

Each function is classified according to its relative functional complexity as low, average or high. The data functions relative functional complexity is based on the number of data element types (DETs) and the number of record element types (RETs). The transactional functions are classified according to the number of file types referenced (FTRs) and the number of DETs. The number of FTRs is the sum of the number of ILFs and the number of EIFs updated or queried during an elementary process. The actual calculation process consists of three steps:
1. Determination of unadjusted function points (UFP)
2. Calculation of value of adjustment factor(VAF)
3. Calculation of final adjusted functional points.

Advantages of FPA:

- Standards are established and reviewed frequently.
- Resulting metrics are logical and straightforward.
- Counting resources are available from requirements stage and applicable for full life-cycle analysis.
- Technology, platform, and language independent.
- Objectively defines software application from the user's perspective.

Drawbacks of FPA:

- Largely a manual process.
- Accurate counting requires in-depth knowledge of standards.
- Some variations exist that are not standardized (Mark II, 3D, full, feature points, object points, etc.).
- Not as much historical data available as SLOC.
- Sometimes backfiring, derived from SLOC can be inaccurate and misleading.

V. SUMMARY

Overall, Cost Estimation models are good for budgeting, tradeoff analysis, planning and control, and investment analysis. As they are calibrated to past experience, their primary difficulty is with unprecedented situations. Table 2 summarizes few activities of various Cost models. [8]

Table2.Activities Covered/Factors Explicitly Considered by Various Cost Models.

Group	Factors	SLIM	PRICE-S	SEER-SEM	COCOMO II
Size Attributes	**Source Instructions**	YES	YES	YES	YES
	Function Points	YES	YES	YES	YES
	OO-related metrics	YES	YES	YES	YES
Program Attributes	**Type/Domain**	YES	YES	YES	NO
	Complexity	YES	YES	YES	YES
	Language	YES	YES	YES	YES
	Reuse	YES	YES	YES	YES
	Required Reliability	?	YES	YES	YES
Computer Attributes	**Resource Constraints**	YES	YES	YES	YES
	Platform Volatility	?	?	YES	YES
Personal Attributes	**Personnel Capability**	YES	YES	YES	YES
	Personnel Continuity	?	?	?	YES
	Personnel Experience	YES	YES	YES	YES
Project Attributes	**Tools and Techniques**	YES	YES	YES	YES
	Breakage	YES	YES	YES	YES
	Schedule Constraints	YES	YES	YES	YES
	Process Maturity	YES	YES	YES	YES
	Team Cohesion	?	YES	YES	YES
	Security Issues	?	YES	YES	NO
	Multisite Development	?	YES	YES	YES
Activities Covered	**Inception**	YES	YES	YES	YES
	Elaboration	YES	YES	YES	YES
	Construction	YES	YES	YES	YES
	Transition and Maintenance	YES	YES	YES	YES

- ✓ A question mark indicates that the authors were unable to determine from available literature whether or not a corresponding factor is considered in a given model.

VI. FUTURE WORK

The future work can further replicate this study for industrial software. We plan to replicate our study to predict effort prediction models based on other machine learning algorithms such as genetic algorithms. We may carry out cost benefit analysis of models that will help to determine whether a given effort prediction model would be economically viable.

REFERENCES:

1. Lee Fischman, Karen McRitchie, and Daniel D. Galorath "**Inside SEER-SEM**", The Journal of Defense Software Engineering April 2005.

2. Dennis Kafura, Salley Henry and Mark Gintner "**The Comparison and Improvement of Effort Estimates from Three Software Cost Models**".

3. Harish Mittal, Pradeep Bhatia "**A comparative study of conventional effort estimation and fuzzy effort estimation based on Triangular Fuzzy Numbers**"

4. Wei Lin Du1, Danny Ho2, Luiz Fernando Capretz 3 "**Improving Software Effort Estimation UsingNeuro-Fuzzy Model with SEER-SEM",** Global Journal of Computer Science and Technology, October 2010.

5. G.H. Subramanian , P.C. Pendharkar and M.Wallace, **—An empirical study of the effect of complexity, platform, and program type on software development effort of business applications",** Empirical Software Engineering, vol. 11, pp. 541-553, Dec. 2006.

6. N.H. Chiu and S.J. Huang, **—** "**The adjusted analogy-based software effort estimation based on similarity distances"** , The Journal of Systems and Software, vol. 80, pp. 628-640, 2007.

7. J. W. Bailey and V. R. Basili, "A **meta model for software development resource expenditure**", in Proceedings of the International Conference on Software Engineering, pp. 107–115, 1981.

8. Barry Boehm, Chris Abts, "**Software Development Cost Estimation Approaches – A Survey",** University of Southern California, Los Angeles, CA 90089-0781.