SOFTWARE COST ESTIMATION

[1] Diana Leonita
[2] Suherman
[3] Dewi Agushinta R

[1, 2] Accounting Information System, dianaleonita@gmail.com
[3] Information System Post Graduate Program, dewiar@staff.gunadarma.ac.id
Gunadarma University

ABSTRACT

Software cost estimation is the process of predicting the effort, schedule and cost required to develop a software system. The fundamental equation for estimating the cost of a software activity is simple in concept, but very tricky in real life. In this paper, we explain how important the software estimate is. Then, we also provide the strengths and the weaknesses of its metrics. There are many estimation techniques and models have been introduced and proposed in software technology, but as a project manager, we must decide what the most appropriate models that we use for are. This paper also provides commercial software estimating tool, COCOMO II that has used by many project managers. The last we also describe about the software risk analysis to give more information to all software estimator. Software cost estimation is used to define the project cost or schedule, to inform investment decisions or to assess whether process or technology improvements are effectives. Even it is difficult to do but we have to face it wisely. In spite of the fact that the software estimates outputs may not always be believed, modern software cost estimating tools are now capable of serving a variety of important project management functions. In addition, the accuracy and precision of such tools is now high enough to merit their use for business agreements such as contracts and outsource agreements.
Keywords: Cost estimation, Effort estimation, COCOMO II

INTRODUCTION

In recent years, software has become the most expensive component of computer system projects. Software also has achieved a bad reputation as a troubling technology. Software projects have tended to have a very high frequency of schedule and cost overruns, quality problems, and outright cancellations. While this bad reputation is often deserved, it is important to note that some software projects are finished on time, stay within their budgets, and operate successfully when deployed (Jones, 2007). It is also the reasons why organizations that get the projects, need to make software effort and cost estimates. Estimation will answer these following questions i.e. how much effort is required to complete an activity, how much calendar time is needed to complete an activity and what is the total cost of an activity.

Project cost estimation and project scheduling are normally carried out together, the cost of development are primarily the cost of the effort involved, so the effort computation is used in both the cost and the schedule estimate. Accurate software cost estimate can be used for generating request for proposals, contract negotiations, scheduling, monitoring and controlling. Underestimating the costs may result in management approving proposed system

that then exceed their budgets, with underdeveloped functions and poor quality, and failure to complete on time. Overestimating may result in too many resources committed to the project or during contract bidding, result in not winning the contract, which can lead to loss of jobs.

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 projects and how well these resources will be used, it can be used to assess the impact of changes and support re planning, projects can be easier to manage and control when resources are better matched to real needs and customers expect actual development costs to be in line with estimates costs.

There is no simple way to make accurate software cost estimate to develop a software system. Although cost and effort are closely related, they are not necessarily related by simple transformation function. Effort is often measured in person-months of the programmers, analysis and project managers. This effort estimate can be converted into a dollar cost figure by calculating an average salary per unit time of the staff involved, and then multiplying this by the estimated effort required.

Many development methods and techniques have been introduced for the last three decades. Each estimation technique has its own strengths and weaknesses. In this paper, we will discuss about the most main publish model/ technique, and some basic terminologies relating them, followed by a discussion of current trend, the implications of this trend, and finally the risk analysis in software cost estimates.

SOFTWARE PRODUCTIVITY

Project manager need productivity estimates to help defined the project cost or schedule to inform investment decisions or to assess whether process or technology improvements are effective. Productivity estimates are usually based on measuring attributes of the software and dividing this by the total effort required for development. There are two types of metrics that have been used (Sommerville, 2004) :

1. Size-related metrics based on some output from the software process. This may be lines of delivered source code, object code instructions, etc.
2. Function-related measures based on an estimate of the functionality of the delivered software.

Function points are the best known of this type of measure. For the size-related metric we would like to discuss about Lines of Source Code (LOC)/ Source Line of Code (SLOC) estimates that can be used to estimate size through analogy – by comparing the new software's functionality to similar functionality found in other historic application.

These are the advantages of LOC/ SLOC :

a. SLOC directly relate to the software to be built, the software can then be measure after completion and compared with your initial estimates.
b. SLOC is easy to count. We can compute SLOC by counting the total number of lines of source of code that are delivered, then divide the count by the total time in programmer-months required to complete the project.

For example :

	Analysis	Design	Coding	Testing	Doc
Assembly Code	3 weeks	5 weeks	8 weeks	10 weeks	2 weeks
High-level language	3 weeks	5 weeks	4 weeks	6 weeks	2 weeks

	Size	Effort	Productivity
Assembly Code	5000 lines	28 weeks	714 lines/ months
High-level language	1500 lines	20 weeks	300 lines/ months

Figure 1. System Development Times

As the best known of function-related measure, function point is not a single characteristic but is computed by combining several different measurements or estimates (Sommerville, 2004). These measurements are external inputs and outputs, user interaction (inquiry types), external interfaces and files used by the system.

The function-point metric takes all of above into account by multiplying the initial function-point estimate by a complexity-weighting factor.

UFC = ∑ (number of elements of given type) X (weight)

Table 1.
Function Point Computation

	Simple	Average	Complex	Total
Inputs	3X	4X[2]	6X[2]	20
Outputs	4X[1]	5X[3]	2X	19
Inquiries	7X	7X	15X	0
Files	5X	10X[1]	17X	10
Interfaces	6X	8X	10X[1]	10
UNADJUSTED FUNCTION POINTS = 59				

The total of these products was adjusted by the degree of complexity based on the estimator's judgments of the software's complexity. Complexity judgments are domain-specific and include factor such as data communications, distributed data processing, performance, transaction rate, on-line data entry, end-user efficiency, reusability, ease of installation, operation, change, or multiple site use. Function points are biased towards data-processing systems that are dominated by input and output operations. For this reason, some people think that function points are not a very useful way to measure software productivity (Furey and Kitchenham, 1997; Armour, 2002 from Sommerville). In short, these are the differences between SLOC and the function point methods :

Table 2.
Source Line of Code Versus Function Point

Source Line of Code	Function Point
Analogy Based	Specification Based
Language Dependent	Language Independent
Design Oriented	User's Oriented
Variation a Function of Languages	Variation a Function of Conventions
Convertible to Function Point	Expandable to Source Line of Code

Cost and Schedule Estimation Techniques

Deciding which of the techniques is the most appropriate for

your program usually depends on availability of data, which is in turns depends on where you are in the life cycle or your scope definition (GSAM Version 3.0).

Of these techniques, the most commonly used is parametric modeling. As mention above, determining which method is most appropriate is driven by the availability of data. Regardless of which method used, a thorough understanding of your software's functionality, architecture, and characteristics, and your contract is necessary to accurately estimate required effort, schedule and cost.

Algorithmic Cost Modeling

Algorithmic cost modeling use mathematical formula to predict project cost based on estimates of the project size, the number of software engineers, products factors, and other process. An algorithmic cost model can be built by analyzing the costs and attributes of completed projects and finding the closest fit formula to actual experience.

An algorithmic cost estimate for software cost can be expressed as (Sommerville, 2004).

$$\text{Effort} = A \times \text{Size}^{B} \times M$$

A : Constant factor that depends on local organizational practices and the type of software that is developed.

Size : Size may be either an assessment of the code size of the software or a functionality estimate expressed in function points.

B : The value of exponent B usually lies between 1 and 1,5.

M : Multiplier made by combining process, product and development attributes, such as the dependability requirements for the software and the experience of the development team.

Most algorithmic estimation models have an exponential component (B) that is associated with the size estimate. This reflects the fact that costs do not normally increase linearly with project size. The larger the system, the larger the value of this exponent.

An algorithmic model has the disadvantages. It is difficult to estimate *size* at an early stage in a project when only a specification is available. And the estimates of the factors contributing to *B* and *M* are subjective. It depends on the estimator's background and experience with the type of system that is being developed.

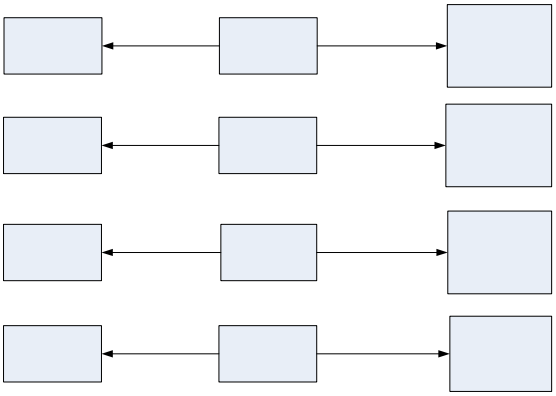
The accuracy of the estimates produced by an algorithmic model depends on the system information that is available. As the software process proceeds, more information becomes available, so estimate becomes more and more accurate. If you use an algorithmic cost estimation model, you should develop a range of estimates (worst, expected and best) rather than a single estimate and apply the costing formula to all of them.

THE CONSTRUCTIVE COST MODEL (COCOMO)

The COCOMO model is an empirical model that was derived by collecting data from large number of software product. This formula link the size of the system and product, project and team factors to the effort to develop the system. The model estimates cost using one of three different development modes: organic, semidetached and embedded (Kelley Cyr, 2007).

Many people, including Sommerville have chosen to use the COCOMO for several reasons, that are well documented, available in the public domain and supported by public domain and commercial tools, it has been

widely used and evaluated in a range of organizations and it has a long pedigree from its first instantiation in 1981 to its most recent version, COCOMO II, Published in 2000.



COCOMO 81 assumed that the software would be developed according to a waterfall process, using standard imperative programming languages such as C or FORTRAN (Sommerville, 2004). While, the COCOMO II supports a spiral model of development and embeds several sub models that produce increasingly detailed estimates. The sub-models that are part of the COCOMO II are in figure 2.

Software Risk Analysis

All software estimating tools typically perform the eight of the nine functions discussed here: 1) Sizing project deliverables, 2) Estimating defect levels and removal efficiency; 3) Selecting project activities; 4) Estimating staffing levels; 5) Estimating effort; 6) Estimating costs; 7) Estimating schedules; 8) Estimating requirements changes during development. The ninth function is not always present in software cost estimation tools: risk analysis. The major risks that need to be analyzed include outright cancellation of the project, the odds of litigation for breach of contract, poor quality control and excessive requirements changes. The software industry has long been troubled by major schedule slippage, major cost overruns, and a high incidence of outright failure.

Table 3.
Software Project Outcomes By Size of Project

	Probability of Selected Outcomes				
	Early	On-Time	Delayed	Canceled	Sum
1 FP	14.68 %	83.16 %	1.92 %	0.25 %	100 %
10 FP	11.08 %	81.25 %	5.67 %	2.00 %	100 %
100 FP	6.06 %	74.77 %	11.83 %	7.33 %	100 %
1000 FP	1.24 %	60.76 %	17.67 %	20.33 %	100 %
10000 FP	0.14 %	28.03 %	23.83 %	48.00 %	100 %
100000 FP	0.00 %	13.67 %	21.33 %	65.00 %	100 %
Average	5.53 %	56.94 %	13.71 %	23.82 %	100 %

(Source : International Thomson Press, 1995)

As can easily be seen from Table 3, small software projects are successful in the majority of instances, but the risks and hazards of cancellation or major delays rise quite rapidly as the overall application size goes up. Indeed, the development of large applications in excess of 10,000 function points is one of the most hazardous and risky business undertakings of the modern world. Of all the troublesome factors associated with software, schedule slips stand out as being the most frequent source of litigation between outsourcing vendors and their clients. Schedule slips are also the main reason for executive frustration with software for internal projects. Fortunately, as of 2008, objective empirical data is beginning to become available in significant quantities. The International Software Benchmark Standards Group (ISBSG)

Results	Percent of Outsource Arrangements
Both parties generally satisfied	70%
Some dissatisfaction by client or vendor	15%
Dissolution of agreement planned	10%
Litigation between client and contractor probable	4%
Litigation between client and contractor in progress	1%

Since the loser of the case may end up paying the legal fees and costs of both sides, the total legal costs can top $12,000,000 or $1,200 per function point. This does not include any fines, damages, or other awards that judges or juries might award to the winning side. Not only are there directs costs for legal fees and expert witnesses, but both parties can expect to lose at least 6 months of productive and effective time on the part of the managers and executives who were involved in the project that is under litigation.

At the end of the day, neither the plaintiff nor the defendant is likely to end up ahead. Litigation is usually a loose-loose situation where neither party gains much of value. This brings up the final point of litigation risk analysis.

Contracts should be clear and unambiguous about four key topics changes to requirements, quality control activities, volumes of delivered defects and progress tracking during development. A litigation and software project failure is an unfortunate by product of poor training and preparation on the part of management on both sides of the case.

CONCLUSION

Software estimating is simple in concept, but difficult and complex in reality. The difficulty and complexity

was founded in 1997. Now that it has been operational for more than 10 years, the volume of measured historical data has reached about 5,000 software projects.

Table 4.
Approximate Distribution of U.S. Outsource Results After 24 Months

required for successful estimates exceeds the capabilities of most software project managers to produce effective manual estimates. The commercial software estimating tools can often out perform human estimates in terms of accuracy, and always in terms of speed and cost effectiveness. However, no method of estimation is totally error-free. The current “best practice” for software cost estimation is to use a combination of software cost estimating tools coupled with software project management tools, under the careful guidance of experienced software project managers and estimating specialists. In addition to normal development estimation, large projects in the 10,000 function point size range should also include specific risk estimates and deal with the problems that are known to cause trouble i.e. estimating accuracy, quality control, change control and status reporting. The strongest point that can be made is that producing excellent software is cheaper and takes less time than producing buggy software that is likely to fail or run late. Producing software that is cancelled or ends up in court will be between 15% and 250% more costly than creating excellent software of the same size and type. To quote Phil Crosby’s famous book, “Quality is Free”. For software, not only is quality free but it costs a great deal less than buggy software and can be produced faster as well.

REFERENCES

[1] Anonymous.1984. *An Approach to Software Cost Estimation*. NASA Goddard Space Flight Center Software Engineering Laboratory. (SEL-83-001) February, 1984.
[2] Anonymous. 2002. *NASA Cost Estimation Handbook*. http://www.jsc.nasa.gov/bu2/NCEH/index.htm. May 2002.
[3] Boehm, B. 1981. *Software Engineering Economics*. Englewood Cliffs. Prentice-Hall, New Jersey.
[4] Boehm, B. 1985. "COCOMO: Answering the Most Frequent Questions," In *Proceedings, First COCOMO Users' Group Meeting.* Wang Institute Tyngsboro. MA. May 1985.
[5] Boehm, et al. 2000. *Software Cost Estimation with COCOMO II*. Prentice Hall. Upper Saddle River. N.J.
[6] Cyr, Kelley. 2009. Basic (Constructive Cost Model) COCOMO. http://cost.jsc.nasa.gov/COCOMO.html. August 2009.
[7] Jones, Capers. 2007. Estimating Software Costs; McGraw Hill, New York.
[8] Sommerville, Ian. 2004. *Software Engineering*. 7th Edition. Addison Wesley. USA.