Running Head: CLASSES OF COST ESTIMATION MODELS

MetaTech Consulting, Inc.

White Paper

Classes of Software Cost Estimation Models with Consideration to Influential Factors

Jim Thomas

August 19, 2003

Abstract

This paper presents a discussion of two classes of software cost estimation models: parametric and heuristic. A brief treatment of each class is followed by a summarization of some of the more prevalent models within the respective class. The parametric class of models is demonstrated through a discussion of the most current version of the Constructive Cost Model (COCOMO 2.0) model and the Function Point Analysis model. An elementary representation of the formulas used for each model is provided to illustrate the most influential factors of each. The bottom-up, top-down, and estimation by analogy models are offered as examples of heuristic cost estimation models. The conclusion summarizes the classes of cost estimation models and illuminates the strengths and weakness of each.

Classes of Software Cost Estimation Models with Consideration to Influential Factors

Effective project management necessitates accurate estimations of time, material, and resources. Estimates are generally needed very early in the development lifecycle though their accuracy is least in the beginning phases. Agarwal et al. (2001) assert that software cost estimation is a continual process present throughout a project lifecycle (Figure 1). By comparing estimates with actual metrics, it is possible to improve the process with each iteration. For nearly fourth years researchers and practitioners have endeavored to synthesize and improve models to produce more accurate and reliable cost estimations for software intensive systems. The present essay draws on some of the most recent research in the field to help illuminate the reader on the maturation of the models being used and the factors that contribute to their accuracy. The purpose of this paper is to convey to the reader in a concise form an understanding of cost estimation models, how and when they are best used, and the factors that practitioners must be aware of to better select from the alternatives.

Accuracy Factors

The available models for cost estimation are each unique and each results in a goodness generally reflected in accuracy and ease of use. Even for the same data set, no two models are likely to result in the same estimate of effort or cost. Doban and Patricza (2001) summarize the errors in estimation as (a) oversimplified models that neglect important information, (b) subjective parameterization, (c) lack of statistically significant number of samples (number of prior similar projects within an organization) to calibrate the model effectively, (d) models that do not represent the state-of-the-art in software development techniques. These factors are closely related to the models and address the shortcomings likely to be experienced in any given environment.

Agarwal et al. (2001) propose a different set of factors affecting accuracy of estimates which includes (a) imprecision and stability of requirements, (b) uniqueness of each project from earlier ones, (c) tendency of software developers to not collect sufficient information on projects, and (d) the fact that estimates are often forced to match the resources that are available. These factors reflect the processes and environment in which the models are used. These conditions each amplify the others to degrade the accuracy and unreliability of the resulting estimates.

Techniques

Cost estimations for software development projects are largely based on the level of effort required to develop the code. As previously mentioned, the level of effort is derived in large part from the *size* of the objective system, its complexity, and a range of other factors. Over the past four decades, researchers and practitioners alike have attempted to improve their ability to estimate these cost factors. Most of the earlier work in the area concerned itself with tangible measurements of size. This type of technique is termed *parametric* method. This is in comparison to the class of *heuristic* techniques that attempt to avoid the troublesome exercise of generating some measurement of size. These two classes of estimation techniques are discussed briefly in subsequent paragraphs. In the interest of brevity, a generalization of each technique will be provided rather than an exhaustive treatment of specific models. A comparative assessment of particular models is reserved for the conclusion of this essay.

Parametric Techniques

Parametric estimation techniques generally require a measurement of the size of the software to be developed. This metric, often measured as some variation of lines of code (LOC), factored together with other performance or environmental metrics are then used to compute a relative or absolute level of effort that can then be used to calculate cost. A true count of size has been demonstrated to provide a reliable basis of estimate for both effort and cost (Musilek, Pedrycz, Sun, and Succi, 2002). Two of the most used models within this class, *Constructive Cost Model (COCOMO 2.0)* and *Function Point Analysis,* are discussed in the following paragraphs.

COCOMO 2.0. This model includes three modules that align well with the classic *waterfall* development lifecycle. The model suitable for estimates required prior to architecture definition is termed the *Application Composition Model* and it produces the most primitive estimates. The *Early Design Model* necessitates additional design details to produce more mature estimate though it does not require the detailed design specification needed by the *Post-Architectural Model*. This last model is the most direct descendent of the COCOMO model though it consists of several improvements in the underlying algorithm.

This technique has as a basis a calculation that determines the number of person-months a project will require based on Equation 1.

$$PM = C * EAF * Size^{B} \tag{1}$$

In this formula, *PM* represents the number of person-months, *C* is a constant of 2.45, *EAF* is the product of seven *effort multipliers*, *size* is the number of thousands of source lines of code, and *B* is the *scaling factor*s that indicate economies of scale. The effort multipliers and the scaling factors are listed in Figure 2.

Musilek, Pedrycz, Sun, and Succi (2002) assert that when using the COCOMO 2.0 model "the most critical independent variable is the projected size of a software system" (p. 13). The authors suggest that, as a means of reducing the impact of the size metric, an estimate may be formed using a fuzzy set that represents a range of size rather than a discrete number. Also, Function Point Analysis (FPA) has been used to estimate code size sufficiently well to input into COCOMO 2.0 by extrapolating from the number and complexity of the products objective functionality as documented in system requirements specification. FPA will be discussed in a later paragraph.

Chulani (2001) illustrated importance of continued calibration of models over time through empirical evaluation of data from completed projects. The Bayesian regression method of calibration used by COCOMO 2.0 allows for the inclusion of *expert judgment* to refine the empirical data. Figure 3 illustrates the improvement of calibration with Bayesian approach (66%) over pure-regression based method (44%) based on a prediction accuracy of 30% and a 161 point set of data.

An extension of the COCOMO model, *Constructive COTS* (COCOTS) is suitable for generating estimates for COTS-based development efforts. This model accounts for costs involved in (a) COTS assessment, (b) component tailoring, (c) *glue code* development, and (d) the volatility of COTS products. It is noteworthy that this model addresses the reality that there is significant coast in building a COTS-based system above the cost of the COTS products themselves. COTS products generally must be configured, optimized, or extended to integrate with the other components of the system. Furthermore, code must be developed to couple together the system components. This integration code, or glue code as it has been termed, represents a non-trivial portion of the overall development activity and must therefore be recognized in the cost estimation process.

Function Point Analysis. Unlike the aforementioned COCOMO model that attempts to estimate the size as a direct count of lines of code, the Function Point Analysis (FPA) model estimates the size of the software project in terms of functional metrics. As they are based on the functional specifications only, the estimations generated by FPA are independent of the targeted development technology.

The origin of the input into the FPA equation (*FP = UAF * VAF*) are the screen formats, report layouts, listing, and specified external interfaces identified in the product specifications. Those items are used to identify the count of (a) external inputs, (b) external outputs, (c) interface processes, (d) internal logical files, and (e) external interface files. Each item in each category is assigned a numerical rating based on perceived complexity. The ratings are then weighted and summed to calculate the *Unadjusted Function Point* count (UFP). A set of general system characteristics are also rated and summed to determine the *Value Adjustment Factor* (VAF). The equation for VAF is provided in Equation 2.

$$VAF = 0.65 + \left[\left(\sum Ci\right)/100\right] \tag{2}$$

Regardless of the platform, language, or hardware used for development, the function point count remains consistent. Considering such a tendency, it is possible to determine if there has been a change in project scope (i.e. scope creep) over the course of the development lifecycle. This can be done by comparing estimate at each phase of development. If there has been change in the function point count, it is reasonable to conclude that requirements have also changed. Through such monitoring, it is possible to assess the effectiveness of the both the initial requirements definition process and the project management process.

Heuristic Techniques

There are a number of techniques within this class. They share the common attribute of basing cost estimates on experience with previous projects. The cost estimations are based on neither coefficients nor environmental variables (Agarwal et al., 2001). A brief characterization of three different heuristic approaches follows.

Bottom-up. This technique can be used following the creation of a comprehensive *Work Breakdown Structure* (WBS). The WBS, identifying each of the discrete software components to be developed, is evaluated by experienced developers who provide estimates of effort for each. This approach assumes that developers have the most reliable sense of complexity of the identified components. Estimates are evaluated to determine reasonableness then are summed to calculate the total project effort. As developers are generally not conscious of holistic program activities, costs related to integration, configuration management, and quality control may by overlooked.

Top-down. In contrast to the bottom-up approach, the top-down approach considers the overall costs associated with software development to include configuration management, and quality control. However, as estimations of this type do not include awareness of the number or complexity of the components, this technique provides a less reliable basis.

Estimation by analogy. This technique, having the least measurable basis of estimate, is the most commonly used approach to cost estimation. It relies solely on the intuition of experienced software practitioners who rely on their recollection of prior development efforts. Agarwal et al. (2001, p. 66) make evident the tendency of developers to favor this technique even in cases when metrics of prior development activities was available, because “it was too difficult to access or the expert could not see how the information would help the accuracy of the estimate.”

Conclusions

Software cost estimation models have continued to evolve over the past four decades. Over time, metrics collected from practitioners have contributed to research and has helped to mature the available techniques. An examination of the available research materials indicates that the number of prominent modeling techniques has reduced somewhat over the past decade through what can be described as a best-of-breed selection process. COCOMO 2.0 and Function Point Analysis continue to maintain strong industry use in embedded systems and for programs that employ structured analysis and design techniques. The shift toward object-oriented development models, coupled with a push to build components to support reuse and to integrate COTS products, has prompted further research in alternative cost estimation models that rely not on the size of the objective system but rather on some other measurement such as the number of and complexity of classes or libraries. Still yet, with the availability of reliable models, many organizations continue to rely on the intuition of their experts and employ heuristic models rather than investing the rigor required in the parametric alternatives. Recognizing this, the most sound advice has been offered by Agarwal et al. (2001) in recommending the concurrent use of more than one approach when performing cost estimation for large software development.

References

Agarwal, R., Kumar, M., Yogesh, Mallick, S., Bharadwaj, R. M., Anantwar, D. (2001). Estimating software projects. *ACM SIGSOFT Software Engineering Notes*, *26*, 60-67. New York: AMC Press.

Chulani, S. (2001). Bayesian analysis of software cost and quality models. *Proceedings of the IEEE Conference on Software Maintenance*, 565 - 568. Washington, DC: IEEE Computer Society.

Doban, O., Pataricza, A. (2001). Cost estimation driven software development process. *Proceedings of the 27th EUROMICRO Conference*. Washington, DC: IEEE Computer Society.

Musilek, P., Pedrycz, W., Sun, N, & Succi, G. (2002). On the sensitivity of COCOMO II software cost estimation model. *Proceedings of the Eighth Symposium on Software Metrics*, 13 - 20. Washington, DC: IEEE Computer Society.

Sedigh-Ali, S., Ghafoor, A., & Paul, R. (2003). Metrics and models for cost and quality of component-based software. *Proceedings of the Sixth International Symposium on Object-Oriented Real-Time Distributed Computing*, 149 - 155. Washington, DC: IEEE Computer Society.

Figure 1

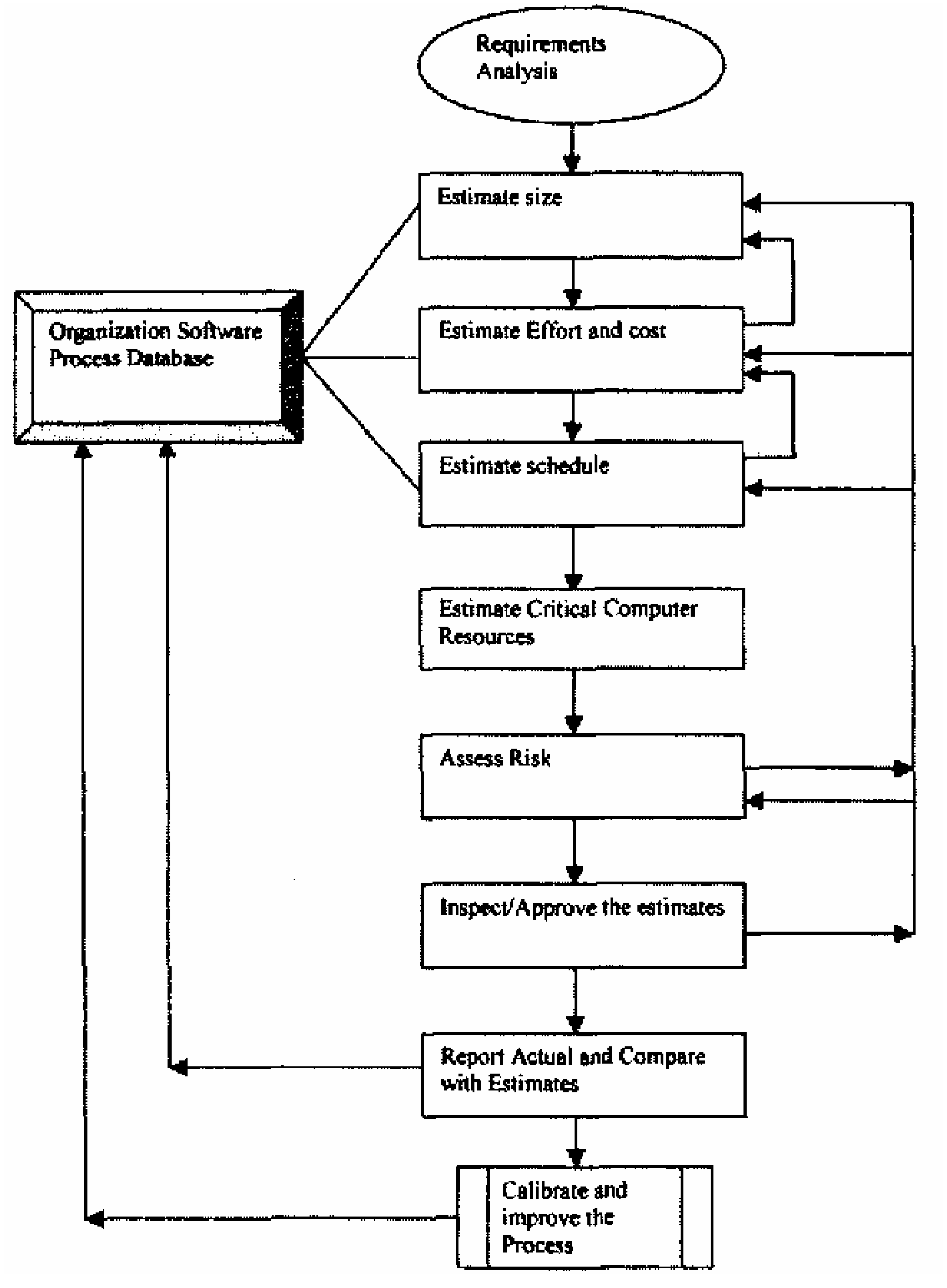
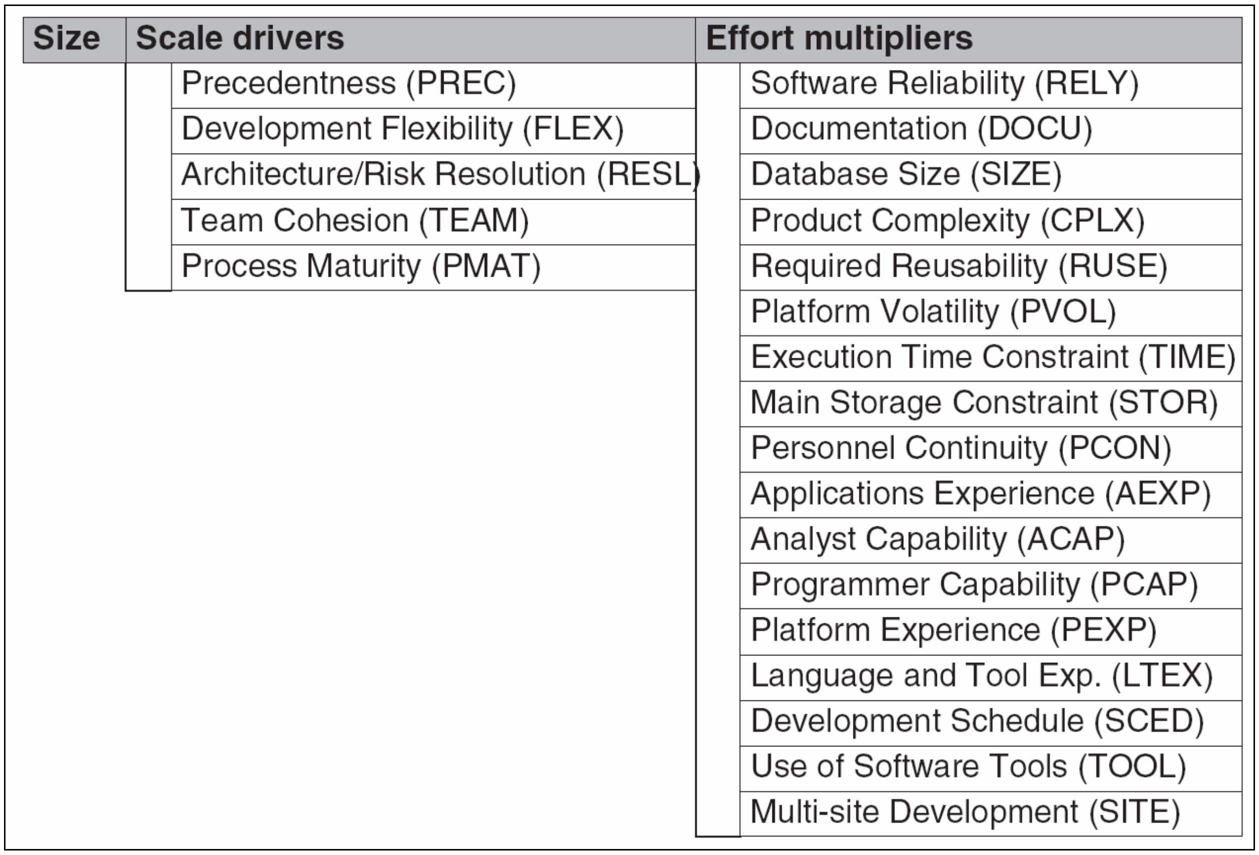

Requirements Analysis
Estimate size
Organization Software Process Database
Estimate Effort and cost
Estimate schedule
Estimate Critical Computer Resources
Assess Risk
Inspect/Approve the estimates
Report Actual and Compare with Estimates
Calibrate and improve the Process

Figure 1. Cost estimation within a project lifecycle process (Agarwal, 2001, p. 60)

Figure 2



Size	Scale drivers	Effort multipliers
	Precedentness (PREC)	Software Reliability (RELY)
	Development Flexibility (FLEX)	Documentation (DOCU)
	Architecture/Risk Resolution (RESL)	Database Size (SIZE)
	Team Cohesion (TEAM)	Product Complexity (CPLX)
	Process Maturity (PMAT)	Required Reusability (RUSE)
		Platform Volatility (PVOL)
		Execution Time Constraint (TIME)
		Main Storage Constraint (STOR)
		Personnel Continuity (PCON)
		Applications Experience (AEXP)
		Analyst Capability (ACAP)
		Programmer Capability (PCAP)
		Platform Experience (PEXP)
		Language and Tool Exp. (LTEX)
		Development Schedule (SCED)
		Use of Software Tools (TOOL)
		Multi-site Development (SITE)

Figure 2. Input parameters in the COCOMO 2.o model (Dobain & Pataricza, 2001)

Figure 3

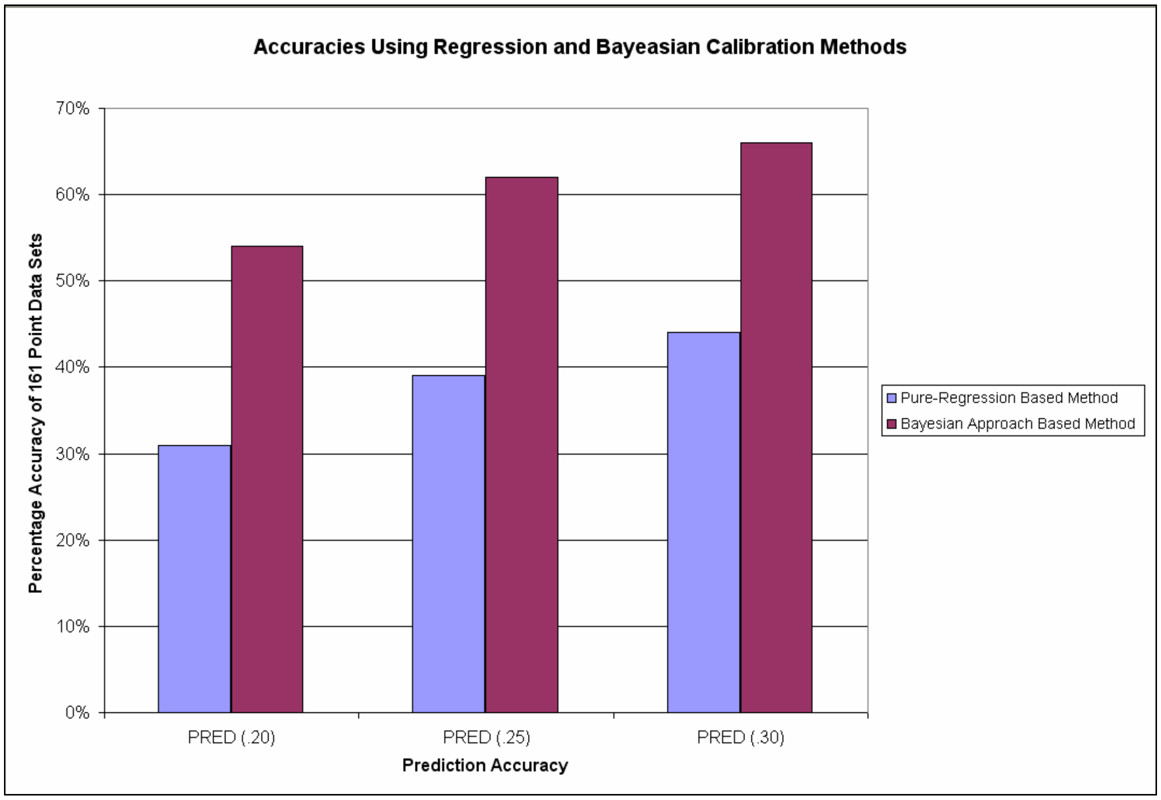


Figure 3. Accuracies using regression and Bayesian calibration methods (Chulani, 2001)