International Journal of Information Technology and Knowledge Management
July-December 2009, Volume 2, No. 2, pp. 437-440

EFFORT ESTIMATION IN COMPONENT BASED SOFTWARE ENGINEERING

Puneet Goswami[*], P. K. Bhatia[**] & Vijender Hooda[*]

Component-based software engineering (CBSE) represents an exciting and promising paradigm for software development. Software components are one of the key issues in CBSE. The software development community is continuously seeking new methods for improving software quality and enhancing development productivity. There is an increasing need for component-based metrics to help manage and foster quality in component-based software development. The traditional software product and process metrics are neither suitable nor sufficient in measuring the complexity of software components, which ultimately is necessary for quality and productivity improvement within organizations adopting CBSE. In the software development process, scheduling and predictability are important components to delivering a product on time and within budget. Effort estimation artifacts offer a rich data set for improving scheduling accuracy and understanding the development process. In this paper, we survey few existing component-based reusability metrics. These metrics give a border view of component's understandability, adaptability, and portability. This paper is split into four sections. First section of this paper emphasizes on need of effort estimation in component based software engineering. Second section of this paper elaborates concept of deriving a metrics using this concept of COCOMO Model. Third section emphasize on how to calculate adjustment factor for the software might be partly developed from software already existing. Fourth section elaborates conclusion and future work. Paper proposes metrics to evaluate effort and development time by conventional and component based software engineering approach and shows the result comparison.

Introduction

Component-based software engineering (CBSE) has recently attracted tremendous attention from both the software industry and the research community. It has been widely recognized that more and more software systems are being built by assembling existing and new components. A lot of research efforts have been devoted to analysis and design methods for component-based software. Although there are many published articles addressing in building component-based programs, hardly very few of them address about component-based measurement [1, 2].

Component-based software development is the process of assembling software components in an application such that they interact to satisfy a predefined functionality [3-6]. Each component will provide and require pre-specified services from other components, thus, the notion of component interfaces becomes an important issue of concern. A key to the success of CBSE is its ability to use software components that are often developed by and purchased from third parties.

Poulin [7] presents a set of metrics used by IBM to estimate the efforts saved by reuse. The study suggests the potential benefits against the expenditures of time and resources required to identify and integrate reusable software into a product. Study assumes the cost as the set of data elements like Shipped Source Instructions (SSI), Changed Source Instructions (CSI), Reused source Instructions (RSI) etc.

Reuse Percentage measures how much of the product can be attributed to reuse and is given as

Product Reuse Percentage = (RSI / (RSI + SSI)) * 100%

Paper proposes several other reusability metrics in terms of cost and productivity like Reuse cost avoidance, Reuse value added and Additional development cost, which can be used significantly for business applications.

Cho *et. al.* [8] proposes a set of metrics for measuring various aspects of software components like complexity, customizability and reusability. The work considers two approaches to measure the reusability of a component. The first is a metric that measures how a component has reusability and may be used at design phase in a component development process. This metric, Component Reusability (CR) is calculated by dividing sum of interface methods providing commonality functions in a domain to the sum of total interface methods. The second approach is a metric called Component Reusability level (CRL) to measure particular component's reuse level per application in a component based software development. This metric is again divided into two sub-metrics. First is $_{\text{CRL LOC}}$, which is measured by using lines of code, and is expressed as percentage as given as

CRL LOC (C) = (Reuse (C) / Size (C)) * 100%

[*] Dronacharya College of Engineering Gurgaon, Haryana, INDIA
[**] Reader, Deptt. of Computer Sc. & Engg, GJUS&T, Hisar, Haryana, INDIA

The second sub-metric is $_{CRLFunc}$, which is measured by dividing functionality that a component supports into required functionality in an application. This metric gives an indication of higher reusability if a large number of functions used in a component. However, the proposed metrics are based on lines of codes and can only be used at design time for components.

Effort estimation consists in predict how many hours of work and how many workers are needed to develop a project. The effort invested in a software project is probably one of the most important and most analyzed variables in recent years in the process of project management. The determination of the value of this variable when initiating software projects allows us to plan adequately any forthcoming activities. As far as estimation and prediction is concerned there is still a number of unsolved problems and errors. To obtain good results it is essential to take into consideration any previous projects. Estimating the effort with a high grade of reliability is a problem which has not yet been solved and even the project manager has to deal with it since the beginning.

Several methods have been used to analyze data, but the reference technique has always been the classic regression method. Therefore, it becomes necessary to use some other techniques that search in the space of non linear relationship. Some works in the field have built up models (through equations) according to the size, which is the factor that affects the cost (effort) of the project the most [Do l00], [KT 85]. The equation that relates size and effort can be adjusted due to different environmental factors such as productivity, tools, complexity of the product and other ones. The equations are usually adjusted by the analyst to fit the real data.

From this perspective, different equation patterns have come out [Do l00], [Hu 97]. But none of them has produced enough evidence to be considered the definitive cost function, in case there is one. Nevertheless, the characteristic that has to be satisfied by the estimation equation is: the model should be capable of doing its best on estimating reliably the majority of the real values.

It hasn't been possible until now to obtain an equation, set of equations or patterns of equations that can satisfy this premise, and therefore there is no reference of comparison parameter. Then it can be assumed that the equations are not a good tool to obtain an optimum prediction.

The estimation of the effort invested in the development of software projects can turn into a complicated problem to be solved if the appropriate models are not available. Unfortunately, until this moment this is the situation, since there are not the necessary records in the software development companies. Years of investigation are required in order to obtain the volumes of information needed to carry out a prediction with a good level of reliability and with a low error margin.

In CBSD, a programmer can be working on multiple components at any given time. Conversely, at any given time a component may have multiple developers. During a single time unit, the project can have multiple components under development by multiple programmers (11)

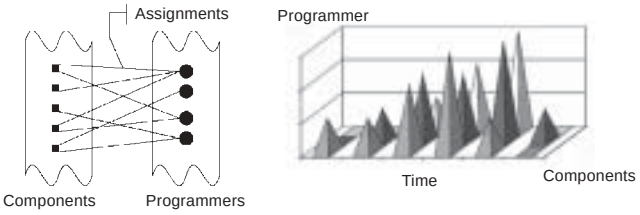

Figure 1: Concept of Components and Time Units of Programmers

Using this concept of components, time units and programmers, Randy K. Smith (11) derive a suite of metrics that characterizes the effect of scheduling on CBSD. These metrics provide a starting point to examine effort estimation in CBSD. The metrics capture the dynamic nature of component development that distinguishes the paradigm from traditional software development.

In order to determine the application of the metrics to effort estimation, research questions must be answered.

COCOMO is a well-studied and accepted effort estimation model. By calculating adjustment factor and size (equivalent) we can calculate effort required for any project in component based software engineering. As the software might be partly developed from software already existing (that is, re-usable code), a full development is not always required. In such cases, the parts of design document (DD%), code (C%) and integration (I%) to be modified are estimated. Then, an adjustment factor, *A*, is calculated by means of the following equation :

The size equivalent is obtained by

$$S \text{ (equivalent)} = (S \times A) / 100$$

Where *S* represents the thousands of lines (KLOC) of the module.

$A = 0.4$ DD + 0.3 C + 0.3 *I* (adjustment factor)

Empirical Study

Intermediate COCOMO equations

$E = a_i(KLOC)^{d_i} *$ EAF

$D = c_i(E)^{d_i}$

Where *E* is effort applied in Person-Months, and *D* is the development time in months. EAF = The multiplying factors for all 15 cost drivers are multiplied to get effort adjustment factor.

Intermediate COCOMO Model

Boehm introduced an additional set of 15 predictors called cost drivers in the intermediate model to take account of the software development environment. Cost drivers are used to adjust the nominal cost of a project to the actual project environment, hence increasing the accuracy of the estimate.

Cost Drivers

(i) Product attributes
   a) complexity of the product
   b) Size of application database
   c) Required s/w reliability

(ii) Hardware Attributes
   a) Run time performance constraints
   b) Memory constraints
   c) Virtual machine volatility
   d) Turnaround time

(iii) Personal Attributes
   a) Analyst capability
   b) Programmer capability
   c) Application experience
   d) Virtual m/c experience
   e) Programming language experience

iv) Project Attributes
   a) Modern programming practices
   b) Use of software tools
   c) Required development Schedule

Consider the effort estimation for a new project with estimated 400 KLOC embedded system. Project manager has a choice of hiring from two pools of developers: Very highly capable with very little experience in the programming language being used.

Developers of low application experience but a lot of experience with the programming language. What is the impact of hiring all developers from one or the other pool?

Cost Drivers	RATINGS					
	Very Low	Low	Nominal	High	Very High	Extra High
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	–
AEXP	1.29	1.13	1.00	0.91	0.82	–
PCAP	1.42	1.17	1.00	0.86	0.70	–
VEXP	1.21	1.10	1.00	0.90	–	–
LEXP	1.14	1.07	1.00	0.95	–	–

Figure 2: Multipliers of Different Cost Drivers

Table 1
COCOMO Modes

Project	a*i*	b*i*	c*i*	D*i*
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Conventional Approach:

This is the case of embedded mode and model is intermediate COCOMO.

$$E = a_i(KLOC)^{d_i}$$

$$= 2.8\ (400)1.20 = 3712\ PM$$

Case I: Developers are very highly capable with very little experience in the programming being used.

EAF = 0.86 × 1.14 = 0.9804

E = 3712 × .9804 = 3639 PM

D = $2.5\ (3639)^{0.32}$ = 34.5 M

Where 0.86 & 1.14 are predefined values by Boehm for personnel attributes cost drivers.

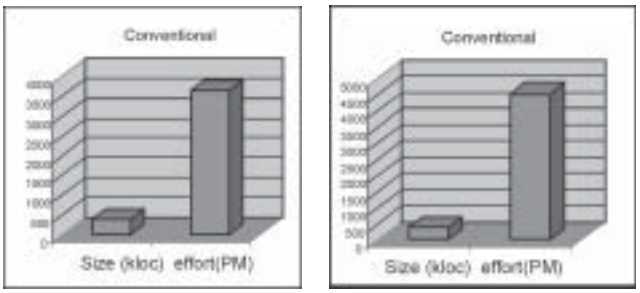

Figure 3: Effort and Development Time Estimation in Conventional Approach

Case II: Developers are of low application experience but lot of experience with the programming language being used.

EAF = 1.29 × 0.95 = 1.22

E = 3712 × 1.22 = 4528 PM

D = 2.5 (4528) 0.32 = 36.9 M

Case II requires more effort and time. Hence, low quality developers with lot of programming language experience could not match with the performance of very highly capable developers with very litter experience.

Component Based Software Engineering Approach:

Size Equivalent

As the software might be partly developed from software already existing (that is, re-usable code), a full development is not always required. In such cases, the parts of design

document (DD%), code (C%) and integration (I%) to be modified are estimated. Then, an adjustment factor, A, is calculated by means of the following equation.

The size equivalent is obtained by

S (equivalent) = (S × A) / 100

Where S represents the thousands of lines (KLOC) of the module.

A = 0.4 DD + 0.3 C + 0.3 I

Considering the DD% =5, C% =10, I% = 5; in the above project then

A = 0.4 * 5 + 0.3 * 10 + 0.3 * 5

A = 2.0 + 3 + 1.5 = 6.5

NOW,

S (Size equivalent) = S* A /100 = 400*6.5/100 = 26

$E = a_i$ (KLOC)^{d_i} * EAF

Case I: Developers are very highly capable with very little experience in the programming being used.

E = 2.8 (26) $^{1.2 *}$ 0.9804

E = 136.93 PM

D = 2.5 (136.93)$^{0.32}$ = 12.06 M = 12 M

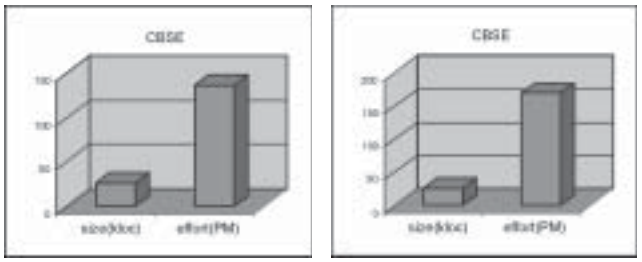

Figure 4: Effort and Development Time Estimation in CBSE Approach

Case II: Developers are of low application experience but lot of experience with the programming language being used.

E = 2.8 (26)$^{1.2}$ * 1.22

E = 170.40 PM

D = 2.5 (170.4) $^{0.32}$ = 12.94 M = 13 M

CONCLUSIONS AND FUTURE DIRECTIONS

The proposed metric appears to be logical and fits the intuitive understanding but is not the sole criteria for deciding the complexity size of the software component on the basis of the computed value of this metric. More work towards the validation of this metric is suggested as future directions by taking into consideration several software components from the software organizations adopting CBSE. This paper identifies and quantifies parameters that impact development effort in CBSD. The parameters identified in this paper specifically examine the characteristics of CBSD. This fundamental paper lays the foundation to examine the dialog of the differences between CBSD and traditional development practices.

REFERENCES

[1] Gill N. S., Grover P. S. Component-based Measurement: Few Useful Guidelines. *ACM SIGSOFT SEN* **28**(6) (2003) 30.

[2] Brereton, B., Budgen, D. Component-Based Systems: A Classification of Issues. *IEEE Computer,* ((2000) 54-62.

[3] Hevner A. R. Phase Containment Metrics for Software Quality Improvement. *Information and Software Technology.* **39** (1997) 867-877.

[4] Gill N. S. Reusability Issues in Component-Based Development. *ACM SIGSOFT SEN,* **28**(6) (2003) 30.

[5] Kamiya, T., Kusumoto S., Inoue K., Mohri Y. Empirical Evaluation of Reuse Sensitiveness of Complexity Metrics. Information and Software Technology. **41** (1999) 297-305.

[6] Sedigh-Ali, S., Ghafoor, A., Paul, Raymond A. Metrics-Guided Quality Management for Component-Based Software Systems. Proceedings of the 25th Annual (COMPSAC'01) (2001). *http://dlib2.computer.org/conferen/compsac/1 37 2/pdf/1 3720303 .pdf.*

[7] Sedigh-Ali, S., Ghafoor, A., Paul, Raymond A. Software Engineering Metrics for COTS-Based Systems. *IEEE Computer,* (2001) 44-50.

[8] J. Poulin, J. Caruso and D. Hancock, "The Business Case for Software Reuse, *IBM Systems International Computer Software and Applications Conference Journal,* **32**(40) (1993) 567-594.

[9] Eun Sook Cho *et al., "Component Metrics to Measure Component Quality",* Proceedings of the Eighths Asia-Pacific Software Engineering Conference, 1530-1362/01.

[10] Brown, Alan W., Wallnau, Kurt C. The Current State of CBSE. *IEEE Software,* (1998) 37-46.

[11] Fenton, Norman E., Neil Martin. Software Metrics: Success, Failures and New Directions. *The Journal of Systems and Software.* **47** (1999) 149-157.

[12] Effort Estimation in Component-Based Software Development: Identifying Parameters Randy K. Smith The University of Alabama P. O. Box 870290 Tuscaloosa, Al 35487-0290 *rsmith@cs.ua.edu.*