

## An approach to estimate software costs within a narrow context

Ngoc-Bao Nguyen, Viet-Ha Nguyen

*College of Technology*

*Vietnam National University, Hanoi*

*Email: {baonn, hanv}@vnu.edu.vn*

This paper proposes a new approach to estimate software costs using case-based reasoning. In this approach, the costs of a new software project are estimated by firstly retrieving the similar previous project and then adapting its costs to the current conditions. The approach focus on the estimation within a narrow context. The project is described as an ontology which allows the managers to estimate with various level of requirement analysis during the development. Moreover, the statistical analysis of previous works (i.e. the COCOMO model) are utilized to reflect the software development domain knowledge.

*Keywords:* software project management, cost estimation, case-based reasoning, ontology, COCOMO

### 1. Introduction

Early and accurate cost estimation is an important factor influencing the profitability of every industry, including software development. However, the task of cost estimation in software development is different from that in other fields due to some particular characteristics of software: First, software is an intellectual product, and thus hard to be thoroughly understood and represented. Second, because of the continual emergence of new technologies, it is too complex to define a universal metric to calibrate all kinds of software. Those characteristics of *a weak theory domain* make software costs difficult to be estimated. In fact, the software cost estimation task will never be an exact science.<sup>1</sup>

Over the years, a number of software cost estimation models have been proposed in attempts to minimize the errors of the estimation.<sup>1,2</sup> However, most of them are based on mathematical models (i.e. statistical models), which seem too theoretical and rigid. Moreover, the proposed models are only available in late phases of the development when requirement analysis is well-defined. Consequently, many project managers find those models

hard to be directly applied to their real works.

In this paper, we propose a new approach to estimate software costs using Case-Based Reasoning (CBR)<sup>3</sup>- a problem solving paradigm particularly suitable for weak theory domains. In this approach, the costs of a new software project are estimated by firstly retrieving similar previous project, and then adapting its costs to the current context. The CBR approach is attractive since there are evidences that experts can perform acceptable estimation basing solely on their specific experiences (i.e. expert judgment).<sup>4</sup>

We believe that the CBR approach is mostly effective in a narrow context, thus our approach is particularly designed for estimating within a certain software developing environment (e.g., the scope of a software company). The project is represented as an ontology to give the managers the flexibility in estimation with various level of requirement analysis. Moreover, the domain knowledge is reflected in the estimated results by utilizing the analysis of software development derived from existent statistical models.

The rest of this paper is organized as follows: In section 2, we summarize some key estimation methods used in software industry. Section 3 presents our proposed approach to estimate software costs using case-based reasoning. The approach is illustrated by some examples given in section 4. Section 5 provides some discussion and considers related works in the fields. In section 6, we summarize our findings and suggest directions for future research.

## 2. Background to software estimation

There are a number of software estimation methods which have been developed and validated in literature as well as in industry. Generally, the conventional methods can be divided into two main categories: *decomposition techniques* and *empirical models*.<sup>1</sup>

Decomposition techniques mean splitting the process or the software itself into small pieces for estimation and then adding the component estimates up to form the overall results. Although those methods are in somewhat similar to the way people deal with complex problems, the demand of decomposition implies that the techniques cannot be applied early. Furthermore, in software development, the costs of system level activities such as integration and document are usually considerable and difficult to be predicted. As a result, decomposition approaches usually lead to underestimated costs.

Empirical models, on the other hand, build a general software model and

estimate the software projects as a whole. Boehm, *et al.*<sup>2</sup> identified several current empirical software estimation models including PRICE-S,<sup>5</sup> SLIM,<sup>6</sup> COCOMO II<sup>7</sup>. . . Most of them were based on mathematical functions with a general form of  $E = A + B \times (ev)^C$ , where  $E$  is the estimation results;  $A$ ,  $B$ ,  $C$  are coefficients derived from regression analysis of historical data and  $ev$  is the estimation variable (i.e. size in SLOC or FP).<sup>1</sup> Those models take the advantages of consistency and objectiveness. Unfortunately, due to the extreme variety in software development, a model derived statistically from a context cannot be useful in others without any calibration to local environment. Although some adjustment techniques (see<sup>7-9</sup>) were added to avoid that situation, they are too complex and difficult for practitioners (as well as customers) to understand and manipulate. In addition, like decomposition techniques, the need of detailed data (e.g., size) in empirical models also prevents the users from flexibly estimating.

More recently, attention has been turning to a variety of machine learning approaches which are trained on local data to estimate the software costs. Srinivasan used an inductive learning system to produce a set of rules for estimating.<sup>10</sup> Dolado applied a genetic programming (GP) approach to investigate the size-effort relationship and build dynamic software process equations.<sup>11</sup> Gary estimated by constructing a back propagation neural network.<sup>12</sup> However, such methods require an extremely large yet convergent historical data. Furthermore, the estimation are incoherent and lack of explanatory value.

### 3. Costs estimating using CBR

Case-Based Reasoning (CBR) is a problem-solving method first appeared in the work on dynamic memory of Schank.<sup>3</sup> In this section, we make use of the CBR idea to propose a new approach for software costs estimation. In this approach, the costs of a project are estimated by adapting the costs of a similar project which has been completed. The approach can be flexibly used during the development and particularly applied within the scope of an organization.

#### 3.1. *The estimating framework*

The estimation framework in our approach is built based on the CBR process proposed by Aarmodt and Plaza.<sup>13</sup> It is described as a cycle of four steps as shown in figure 1.

The costs of a new project (i.e. a case) are estimated by firstly *retriev-*

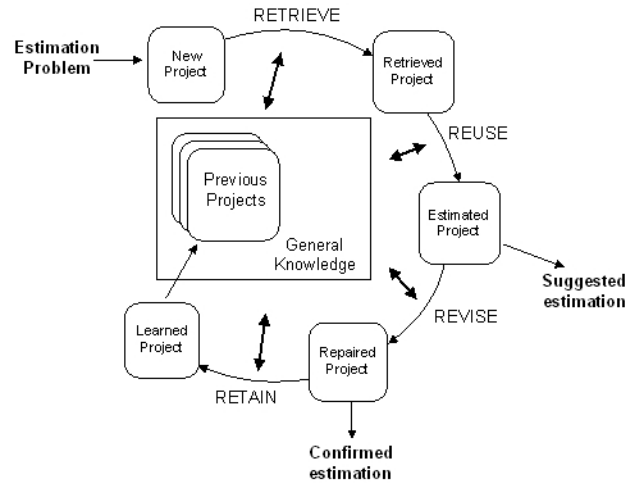


Fig. 1. The estimating framework.

*ing* the most similar project from a set of previous ones. Then, the known costs of this project are *reused* by adapting to the circumstance of the estimating one. The *revise* step evaluates, normally by human, the estimation suggested previously whether it fits the real world environment. The last step is *retaining* where completed projects are stored into the knowledge base for future uses. All the four steps may be supported by the domain knowledge of software development to improve their performance.

In the followings, we will describe in more details three activities of our estimation approach, which are the project representation, the retrieval of similar project, and the adaptation of previous costs.

### 3.2. Project representation

As the life cycle processed, the requirements of a project become more and more well-defined. We represent projects by an ontology, an explicit specification of a shared conceptualization;<sup>14</sup> so that the estimation can be performed at different levels of the requirement analysis.

Figure 2 shows the details of the project ontology representing a software project in our approach. In this figure, [PROJECT] is the top level concept consisting of two sub-concepts: [COSTS] and [COST DRIVERS]. The [COSTS] represents a set of values managers desire to estimate while [COST DRIVERS] represents factors (named cost drivers) believed to influence those values. There are a lot of factors may influence the final costs;<sup>1,2,15</sup> yet since our

goal is to estimate within a specific circumstance, we just account for the features of the product (i.e. the software itself), not the features of the developing environment.

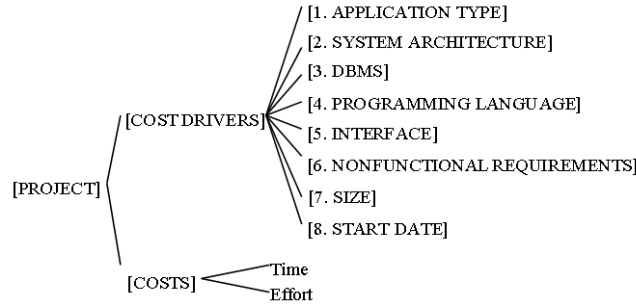


Fig. 2. The project ontology.

Each cost driver is again considered as a concept which is classified to several sub-concepts and instances in a hierarchical structure. Figure 3 illustrates an example of the ontology of cost driver [PROGRAMMING LANGUAGE] where elements in upper case present sub-concepts and elements in lower case present instances.

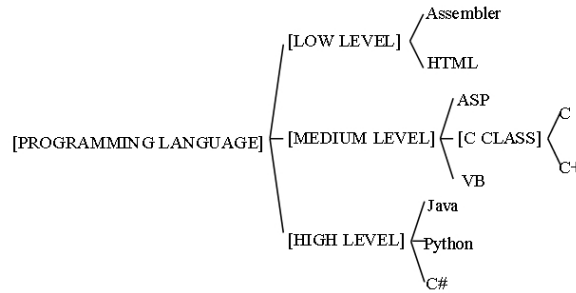


Fig. 3. The ontology of a cost driver.

Previous studies<sup>16,17</sup> suggested that to obtain a reasonable estimation, at least a size factor should be taken into account. However, most of past approaches tend to use the same sizing model during their estimation. In this work, depending on the details of the requirement analysis, the size of

a project can be flexibly considered in different levels of sizing models (e.g., User Functional Requirements, Function Points, SLOC...).

### 3.3. Retrieval

The aim of retrieval is to extract the nearest project from the historical database. To indicate which project is the nearest, we define the following similarity metric.

In the estimating process, some cost drivers may be not defined yet. Despite of that, the project similarity can still be calculated basing on the other available cost drivers. We use an weighted average function to calculate the similarity between two projects:

$$SIM(T, S) = \frac{\sum_{i=1}^n |sim(T_i, S_i)| \times w'_i}{\sum_{i=1}^n w'_i}, \quad (1)$$

where  $SIM(T, S)$  is the similarity between projects  $T$  and  $S$ ;  $sim(T_i, S_i)$  is the similarity of their cost driver  $i$  and  $w'_i$  are a *extended weight* determined by:

$$w'_i = \begin{cases} w_i, & \text{if cost driver of two projects are defined;} \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

where  $w_i$  is the weight indicating the significance of cost driver  $i$ .

The similarity between two concepts is:

$$sim(T_i, S_i) = \frac{\sum_{j=1}^m \sum_{k=1}^n sim(t_j, s_k)}{mn}, \quad (3)$$

where  $t_j$  and  $s_k$  are all instances, either directly or indirectly, under concepts  $T_i$  and  $S_i$  of cost driver  $i$ .

Likewise, the similarity between a concept and an instance is defined as:

$$sim(T_i, s_i) = \frac{\sum_{j=1}^m sim(t_j, s_i)}{m}, \quad (4)$$

where  $t_j$  are all instances under concept  $T_i$ .

The similarities between instances are calculated basing on the characteristics of individual cost drivers in software development.

The cost drivers No. 1-6 belong to categorical types and by now, the similarity of its instances is determined by referring to a similarity table.

The values in this table are pre-defined basing on software engineering knowledge and normalized to be in the interval  $[-1, 1]$ . We are studying some fuzzy-based approaches to improve the similarity calculations for categorial cost drivers but they are out of the scope of this paper.

The similarity function of size is determined by:

$$sim(t_i, s_i) = \left[ \frac{1}{2} \left( \frac{t_i}{s_i} + \frac{s_i}{t_i} \right) \right]^{-\alpha}, \quad (5)$$

where  $t_i$  and  $s_i$  are size of two project in the same sizing model,  $sim(t_i, s_i)$  is the similarity between them and  $\alpha$  ( $\alpha > 0$ ) is a scale factor .

In software engineering, the developing environment as well as the technologies are rapidly changed. The old projects may have little meaning in the estimation of the new one. Thus, we use an exponential form to present the similarity of start date:

$$sim(t_i, s_i) = \beta^{-\frac{t_i}{s_i} |t_i - s_i|}, \quad (6)$$

where  $\beta$  ( $\beta > 1$ ) indicates the growth rate in the software industry.

Although the similarity metric above seems to have so many degrees of freedom, since our calculations are just used for selecting the similar project from a limit number of the given ones, we believe that a more delicate metric would not give more accurate results though it complicates the estimating process.

### 3.4. Adaptation

We make use of the analysis derived from statistical estimation models to adapt the costs of the retrieved project. Particularly, the previous costs are adapted by COCOMO-like functions:

$$Time_{current} = aR^b \times Time_{previous} \quad (7)$$

$$Effort_{current} = cR^d \times Effort_{previous} \quad (8)$$

where  $a, c$  are the differential coefficients of the project multiplicative adjustment factors;  $b, d$  are the exponential scales of diseconomy <sup>a</sup>;  $R$  is a size differential coefficient.

<sup>a</sup>the terms are used according to the COCOMO II model definition.<sup>7</sup>

Since the current and the retrieved project share some common features, we assume that the differences of the project multiplicative adjustment factors can be inferred from the differences of non-functional requirements. Thus, we use a non-functional requirements differential coefficient as a single representative, which is defined as:

$$\delta = 1 + \text{sim}(T_i, S_i), \quad (9)$$

where  $\text{sim}(T_i, S_i)$  is the similarity of the non-functional requirements.

The scale exponents  $b$  and  $d$  reflect the characteristic of the organization and calculated basing on the analyses of the COCOMO II model.

$$b = 1.01 + 0.01 \sum w_i \quad (10)$$

$$d = (0.33 + 0.2 \times (b - 1.01)) \times b \quad (11)$$

where  $w_i$  are rated according to table 1.

Finally, the size differential coefficient is determined by:

$$R = \frac{t_i}{s_i}, \quad (12)$$

where  $t_i$  and  $s_i$  are size of two projects measured in the same sizing model.

Table 1. Rating scheme for the COCOMO II scale factors

Scale Factors ( $w_i$ )	Very Low(5)	Low(4)	Nominal(3)	High(2)	Very High(1)	Extra High(0)
Precedentedness	thoroughly unprece- dented	largely unprece- dented	somewhat unprece- dented	generally familiar	largely familiar	thoroughly familiar
Development Flexibility	rigorous	occasional relax- ation	some relax- ation	general confor- mity	some confor- mity	general goals
Architecture/ risk resolution	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
Team cohesion	very dif- ficult in- terac- tions	some dif- ficult in- terac- tions	basically coopera- tive inter- actions	basically coopera- tive inter- actions	basically coopera- tive in- terac- tions	seamless interac- tions
Process matu- rity	Weighted average of "Yes" answers to CMM Maturity Questionnaire					

#### 4. Examples

Assume that we are estimating a project  $P$  with a knowledge base of 3 projects  $P1, P2, P3$  in table 2. At an early state of the development, the size of  $P$  is only available in Number of User Functional Requirements (UFR) and the interface has not been defined yet. The weights of each cost driver in figure 2 are assigned to the value of  $\{10, 2, 4, 5, 1, 4, 5, 10\}$ , respectively. The constants in equation (5) and (6) are chosen as common values of 2.00 and 1.67. Then the similarity between  $P$  and  $P1, P2, P3$  are calculated as shown in table 2.

Since  $P2$  is the nearest project to  $P$ , it is chosen for adaptation. We assign 1.05 to the scale exponent  $b$  in equation (10) and derive the other exponent  $d$  in equation (11) as 0.35 (i.e. the constants of the typical COCOMO organic mode<sup>15</sup>). The size and non-functional requirements differential coefficient are calculated as  $R = 0.80$  and  $\delta = 1.40$ . Then, the estimated results would be  $Time = 18.13$  and  $Effort = 221.51$ .

At later states when the requirement analysis of  $P$  is refined, we know more exactly that the Programming Language is ASP, the interface is web-based and the size can be determined in Function Point as 14 FPs. Then, the similarities are recalculated as shown in table 3. In this case, the nearest project is changed to  $P1$ . Using the same calculations as above, we will obtain a new estimation as  $Time = 16.40$  and  $Effort = 234.39$ . Hopefully, when the more detailed information is available, the more "precise" project is retrieved and the estimated results are accordingly more reliable.

Table 2. An example of costs estimation in early states

Project factors	$P$	$P1$	$(P_i, P1_i)$	$P2$	$(P_i, P2_i)$	$P3$	$(P_i, P3_i)$
App Dom	man	library	0.70	store	0.70	net	0.50
Sys Arch	s.alone	dist	0.80	s.alone	1.00	c/s	0.40
DBMS	simple	My SQL	0.80	Access	0.80	none	0.40
Prog Lang	med	ASP	0.80	VB	0.80	Java	-0.40
Interface	undef	web	0.00	graphic	0.00	web	0.00
NF Reqs	med	low	0.60	low	0.60	high	-0.40
Size	4UFR	5UFR/ 15FP	0.95	5UFR/ 18FP	0.95	3UFR/ 8FP	0.92
Start Date	03/06	03/05	0.60	06/05	0.68	03/05	0.60
Proj Sim			0.71		0.74		0.53
Time		12.00		14.00		8.00	
Effort		180.00		200.00		160.00	

Note:  $(P_i, P_{x_i})$  indicates the similarity of cost driver  $i$  between project  $P$  and  $P_x$

Table 3. An example of costs estimation in later states

Project factors	$P$	$P1$	$(P_i, P1_i)$	$P2$	$(P_i, P2_i)$	$P3$	$(P_i, P3_i)$
Prog Lang	ASP	ASP	1.00	VB	0.60	Java	-0.50
Interface	web	web	1.00	graphic	0.50	web	1.00
Size	4UFR/ 14FP	5UFR/ 15FP	0.99	5UFR/ 18FP	0.94	3UFR/ 8FP	0.86
Proj Sim			0.76		0.72		0.56

## 5. Discussion and related works

The previous statistical methods estimate the costs as direct mathematical functions of project parameters; thus they require all factors of project as well as their incoherent relationship must be reveal and formulate. Our approach otherwise uses the project parameters just for approximately comparing and selecting similar projects. For those reasons, it keeps the project representation simple and gives more flexibilities to the estimating process. For instance, although the environmental factors are extremely complex and vague, when estimating within a narrow context those factors implicitly remain constant. Using CBR as a strategy of estimation, we don't have to account for them (though they are still presented in the final results).

It is known that software development is a wide and often-changing domain. To archive an acceptable estimation, most of previous approaches require tuning their model to the local environment.<sup>8</sup> In our approach, such tuning task is automatically performed by a "learning" process where new projects are captured to enrich the project database. The database itself reflects the characteristics of the developing environment, and as it is changed the estimated results will be adapted accordingly.

There has been several works in the area of applying CBR to software cost estimation such as Estor,<sup>18</sup> FACE,<sup>19</sup> ANGEL,<sup>17</sup> F\_ANGEL<sup>20</sup> (an extension of ANGEL using fuzzy similarities). However, all of them use a flat structure for project representation whereas our projects are represented by a layer structure as an ontology. Using such a flexible representation, managers are able to execute the estimation with various level of requirement analysis.

In,<sup>16</sup> Sarah, *et al.* introduced a CBR approach to early software cost estimation. In,<sup>21</sup> Belen, *et al.* presented the CBR<sub>Onto</sub> architecture which combine CBR and Ontolgy ideas. Those models seem work as CBR frameworks where all of case features share a common similarity function (i.e. the Euclidean distance) or the similarity determining tasks is left to the users. In this work, on the other hand, we construct an approach specifically tai-

lored for software development field. The project structure as well as the similarity calculation are predefined based on our studies of the software development. Moreover, the analysis derived from previous statistical models is also utilized in the estimating process. By this way, the domain knowledge of software development is automatically presented in the estimation.

## 6. Conclusion

In this paper, we presented an approach to estimate software costs using case-based reasoning. The approach is particularly used for estimation within a narrow context, for example the scope of an company. It does not require elaborate requirement analysis as some predecessors and can be flexibly applied in various phases of the development. The estimated results are clear and coherent in that they are directly derived from previous projects. Moreover, by concerning specific characteristics of software development, the approach seems to be more “software-oriented” than some other analog-based alternatives.

The current problem in our approach is the cost drivers as well as their similarity calculations were still roughly built. As for future works, those issues should be analyzed more thoroughly with the consideration to some existing standards. Mechanisms of parameter learning and database refining should also be investigated to improve the system performance. Furthermore, we are planning to implement our approach to an application which can be used and validated in real software developing environments.

**Acknowledgement.** This work is partially supported by the National Fundamental IT Research Project.

## References

1. R. S.Pressman, *Software engineering: a practitioner's approach (5th ed.)* (McGraw-Hill, Inc, New York, NY, USA, 2001).
2. B. Boehm, C. Abts and S. Chulani, *Ann. Softw. Eng.* **10**, 177 (2000).
3. C. K. Riesbeck and R. C. Schank, *Inside Case-Based Reasoning* (Lawrence Erlbaum Associates, Inc, Mahwah, NJ, USA, 1989).
4. M.Jørgensen, G. Kirkeboen *et al.*, Human judgement in effort estimation of software project, in *Beg, Borrow, or Steal Workshop, International Conference on Software Engineering*, (Limerick, Ireland, 2001).
5. Park R, The central equations of the price software cost model, in *4th CO-COMO Users' Group Meeting*, 1988.
6. L. H. Putnam and W. Myers, *Measures for Excellence: Reliable Software on Time, within Budget* (Prentice Hall Professional Technical Reference, 1991).

7. B. Boehm, B. Clark *et al.*, *American Programmer*, 2 (1996).
8. M. Baldassarre, D. Caivano and G. Visaggio, *icsm* **00**, p. 105 (2003).
9. M. A. Sicilia, J. J. Cuadrado-Gallego *et al.*, *KYBERNETIKA* **35** (2004).
10. K. Srinivasan and D. Fisher, *IEEE Trans. Softw. Eng.* **21**, 126 (1995).
11. J. J. Dolado, Limits to the methods in software cost estimation, in *The 1st International Workshop on Soft Computing Applied to Software Engineering*, eds. C. Ryan and J. Buckley (Limerick University Press, 1999).
12. G. D. Boetticher, Using machine learning to predict project effort: Empirical case studies in data-starved domains, in *The First International Workshop on Model-based Requirements Engineering*, (San Diego, 2001).
13. A. Aamodt and E. Plaza, *AI Communications* **Vol 7**, 39 (1994).
14. T. R. Gruber, *Knowledge Acquisition*, p. 38 (1993).
15. B. Boehm, *Software Engineering Economics* (Prentice-Hall, 1981).
16. S. J. Delany and P. Cunningham, *The Application of Case-Based Reasoning to Early Software Project Cost Estimation and Risk Assessment*, tech. rep., Department of Computer Science, Trinity College Dublin, TDS-CS- 2000-10 (2000).
17. M. Shepperd and C. Schofield, *IEEE Trans. Softw. Eng.* **23**, 736 (1997).
18. S. Vicinanza, M. J.Pritula and T. Mukhopadyay, Case-based reasoning in software effort estimation, in *The 11th International Conference on Information Systems*, 1990.
19. R. Bisio and F. Malabocchia, Cost estimation of software projects through case based reasoning, in *The First International Conference on Case Based Reasoning*, 1995.
20. A. Idri, A. Abran and T. M. Khoshgoftaar, *Engineering Intelligent Systems* **159**, 64 (2004).
21. B. Diaz-Agudo and P. A.Gonzalez-Calero, An architecture for knowledge intensive CBR systems, in *EWCBR 2000*, (Springer - Verlag, 2000).