An Extension of COCOMO II for the B-Method

Yujun Zheng
College of Computer Information Engineering, Jiangxi Normal University
Nanchang 330027, China
86-10-51556190
uchengz@yahoo.com.cn

Kan Wang
Academy of Armored Force Engineering
Beijing 100072, China
86-10-66719327
hjbf@sina.com.cn

Jinyun Xue
College of Computer Information Engineering, Jiangxi Normal University
Nanchang 330027, China
86-791-8507989
jinyun@jxnu.edu.cn

ABSTRACT
Constructive Cost Model (COCOMO) series of models are widely used for estimation of informal software development effort. To meet the need for estimating size and cost of projects developed with the B-Method, we extend the sub-models of COCOMO II by introducing specification-based size metrics, adjusting relevant coefficients, and narrowing the value ranges of certain cost drivers. Our work is intended to serve as a start point for a general cost model for formal software development processes.

Categories and Subject Descriptors
D.2.9 [Software Engineering]: Management – Cost estimation.

General Terms
Management, Measurement, Economics.

Keywords

1. INTRODUCTION
Constructive Cost Model (COCOMO) series of models [5-8] has been widely used for effort estimation of general (informal) software projects. Emerging COCOMO II extensions recognize different development approaches such as prototyping, component composition and database programming, and provide a family of sub-models to address the issues related to new software life cycle processes and capabilities.

Formal methods use mathematically precise models to build software systems, and therefore offer the promise of significant improvements in software quality and reliability. Recently they have been increasingly applied in industrial software development. It is recognized that the introduction of formal methods into the industry effectively adds to project costs or risks [16], but few researches have been done to establish special cost models for formal development processes.

Base on data collected from 12 projects of software development using the B-Method [1], we derive a cost estimation approach from the constructive cost model (COCOMO) II. Main extensions and adaptations include measuring the software size based on formal specifications instead of code, adjusting relevant coefficients for specification-based size, and narrowing the value ranges of certain cost drivers to fit in with the B-Method and its supporting environment.

Although many believe that formal methods is a reverse from the normal cost model for software development [10], our result shows that basic formulas in COCOMO II can be adapted and extended to reasonably estimate size and cost of software developed with the B-Method. Future work includes making the approach applicable to other formal methods.

2. AN OVERVIEW OF THE B-METHOD
The B-Method is a theory-based, model-oriented formal method for specifying, designing and coding software systems. It provides a common framework for the construction of specifications and refinements at all levels using the abstract machine notation (AMN), a state-based formal specification language in the same school as VDM [13] and Z [17].

The B-Method supports modular structuring based on a collection of structuring mechanisms for information hiding, modularization and the compositionality of module operations, reuse and proof decomposition [12]. Systems are modeled as a collection of interdependent abstract machines, for which an object-based approach is employed at all stages of development. There are three types of machines in the B-Method:

1. MACHINE: represents an (initial) abstract machine that encapsulates state consisting of a set of variables constrained by an invariant, and operations which may change the state and return a sequence of results while maintaining the invariant.
2. REFINEMENT: represents an (intermediate) abstract machine that refines a MACHINE or another REFINEMENT by replacing its state with more concrete state and/or replacing its operations with more concrete operations.
3. IMPLEMENTATION: represents an (ultimate) abstract machine that does not need to be further refined and can be implemented with executable programs or hardware.

The B-Method is supported by the B-Toolkit [2], which provides automatic features including syntax and type checking, proof obligation generation, properties proving, failed proof exploration, symbolic animation, and code generation.
3. EXTENSION OF COCOMO II
COCOMO II contains four sub-models, which are fully described in [6] and [7]. Based on data analysis, we adapt and extend these sub-models to software development with the B-Method.

3.1 The Application-Composition Model
The application-composition sub-model is designed to estimate the effort required for applications composition and prototyping projects. The final formula for computing the estimated person-months (PM) is:

\[ PM = \frac{NAP \times (1 - \% \text{reuse})}{100} \times PROD \]

(1)

NAP is the total number of application points in the delivered system, \%reuse is an estimate of the amount of reused part in the development, and PROD is the application-point productivity rate. In COCOMO II, PROD is suggested to be determined from the scheme shown in Table 1.

Table 1. Application-point productivity in COCOMO II

<table>
<thead>
<tr>
<th>Developer’s experience and capability</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASE maturity and capability</td>
<td>Very Low</td>
<td>Low</td>
<td>Nominal</td>
<td>High</td>
<td>Very High</td>
</tr>
<tr>
<td>Prod (NAP/month)</td>
<td>4</td>
<td>7</td>
<td>13</td>
<td>25</td>
<td>50</td>
</tr>
</tbody>
</table>

We regard the B-Toolkit as a rapid-composition software environment at the level of high maturity and capability. Moreover, in contrast to executable programming languages, abstract specification language improves the overall software productivity significantly, meanwhile narrows the disparities among developers. As shown in Figure 1, the application-point productivity of the projects investigated range from about 27 to 57 NAP/PM.

Furthermore, based on data collected on these projects and data from a training department of the Academy of Armored Force Engineering, the average application-point productivity of a regularly trained B developer is about 45 NAP/PM. According to the rating scheme of COCOMO II, we establish a set of recommendatory productivity levels for the B-Method developers, as shown in Table 2.

Table 2. Application-point productivity for the B-Method

<table>
<thead>
<tr>
<th>Developer’s experience and capability</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prod (NAP/month)</td>
<td>25</td>
<td>35</td>
<td>45</td>
<td>52</td>
<td>60</td>
</tr>
</tbody>
</table>

3.2 The Early Design Model
The early design sub-model is used to make initial effort estimates during the early stages of system design, where the detailed architecture of the system is generally unknown. Most algorithmic cost models employ the following standard formula to produce the estimates:

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

(2)

In the early design sub-model of COCOMO II, the coefficient A is proposed to be 2.94; the component B varies from 1.1 to 1.24 depending on the novelty of the project, the development flexibility, the risk resolution processes used, the cohesion of the development team, and the process maturity; the multiplier M is the product of a set of effort-multiplier cost drivers based on characteristics including product reliability and complexity (RCPX), required reusability (RUSE), platform difficulty (PDIF), personnel capability (PERS), personnel experience (PREX), required development schedule (SCED), and support facilities (FACI):

\[ B = 1.01 + 0.01 \sum_{i=1}^{7} W_i \]

(3)

\[ M = \prod_{i=1}^{7} EM_i \]

(4)

In our extension for the B-Method based development processes, the formulas for determining the component B and the multiplier M also apply, but the RCPX and FACI ratings are generally limited to the range from High to Extra High. Moreover, the size in formula (2) should be expressed in lines of specifications (LOS) instead of source lines of code (LOC). For AMN invariants and substitutions in abstract machine specifications, their equivalent predicates’ each conjunctive or disjunctive component is taken as one line of specification.

According to project data, the average number of LOS to implement an application point varies from about 6 to 14 (see Figure 1. Data distribution for NAP/PM.)
Figure 2), and the coefficient \( A \) varies from 4 to 13. In the early project stages, this coefficient can be roughly set to 8.

3.3 The Post-Architecture Model

When a software life-cycle architecture of the system has been developed, the post-architecture sub-model is used to make more accurate effort estimates. In COCOMO II, the sub-model also employs the formulas (2), (3) and (4) to produce estimates, but an extensive set of 17 multiplicative cost drivers are used in the formula (4).

For the B-Method based development processes, once an initial architectural design is available, the abstract machine framework is known and the system functions has been decomposed and assigned to individual machines. That is, the informal specification for almost every machine to be developed is worked out. In consequence, we can estimate the total size of final specifications more accurately based on the design information about the abstract machine framework of the system. For the three types of abstract machines, their average LOS/per-machine of the projects investigated are shown in Figure 3:

Furthermore, we employ a five-point scale from VERY LOW to VERY HIGH to rate the complexity of each abstract machine, and compute the estimated size in LOS using the following formula:

\[
\text{Size} = \sum_{i=1}^{3} \sum_{j=1}^{5} (C_{ij} \times NAM_{ij})
\]  

(5)

Here \( NAM \) is the number of machines with \( i \)-type and at \( j \)-level of complexity, and \( C \) is the rating number of LOS per machine from the scheme in Table 3:

<table>
<thead>
<tr>
<th>AM complexity</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>MACHINE</td>
<td>10</td>
<td>17</td>
<td>28</td>
<td>50</td>
<td>80</td>
</tr>
<tr>
<td>REFINEMENT</td>
<td>8</td>
<td>15</td>
<td>32</td>
<td>60</td>
<td>120</td>
</tr>
<tr>
<td>IMPLEMENTATION</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>25</td>
<td>45</td>
</tr>
</tbody>
</table>

Table 3. Rating LOS/AM for different AM types

Since the computation of size has taken the complexity of each abstract machine into consideration, only eight multiplicative cost drivers of COCOMO II are used for computing the final effort multiplier: analyst capability (ACAP), programmer capability (PCAP), applications experience (AEXP), personnel experience (PEXP), language and tool Experience (LTEX), personnel continuity (PCON), required development schedule (SCED), and multi-site development (SITE). Their rating levels are as same as those in the post-architecture sub-model of COCOMO II.

3.4 The Reuse Model

The use of formal methods in system development can help to overcome problems and to aid the promotion of software reuse [9]. However, it is difficult to develop a simple linear or nonlinear model to estimate the cost of reusing software in formal development processes. The main reasons include:

1. The fundamental reuse in formal methods is with respect to “models” or “mathematical theories” [11], not with respect to code.
2. For systems with high safety and reliability requirements, proof obligations of reused components always need to be regenerated rather than reused.
3. For software refactoring or reengineering, few parts of the old system can be directly reused in formal development, and the total effort is supposed to be equal to that from scratch.

In the B-Method projects we studied, the benefits of reuse are further difficult to be quantified mainly because:

1. As the basic four reuse mechanisms for constructing large abstract machines from other machines in the B-Method, INCLUDES, USES, IMPORTS and SEES can all be regarded as black-box reuse without understanding and changing: Only the effort of reuse decision-making needs to be considered.
2. The effort for machine refinement reuse, REFINES, has been taken into account in the specification size estimation (of REFINEMENT and IMPLEMENTATION) in the post-architecture sub-model.
3. The B-Toolkit does not encourage the reuse of user defined rules as user theories are specified to the lemma currently being proved, and so cannot be reused for other proofs [4].
Thus, we simply use a formula derived from the reuse sub-model of COCOMO II to calculate the number of equivalent lines of new specifications (ELOS) from the number of lines of specifications to be adapted (ALOS):

\[
ELOS = \frac{ALOS \times (10 + AA + 0.4 \times DM)}{100}
\]  

(6)

Here AA (assessment factor) varies from 0 to 2 depending on the degree of assessment needed for reuse decision-making, and DM represents the percentage of design modification. The understanding factor SU of COCOMO II is replaced by a constant 10, as we believe that mathematics and AMN are sufficiently general and unambiguous for software understanding.

4. CONCLUSION

With the increasing use of formal methods in the industry, organizations need to make effort and cost estimates for formal development processes. Nevertheless, most estimates are based on qualitative techniques such as expert judgments and analogies [5]. There are disputations about whether traditional algorithmic cost models can be adapted for formal methods, or an entirely new model should be developed.

Base on our historical and experimental project data, we extend the sub-models of COCOMO II to fit in with the B-Method by making specification-based size estimates, adjusting relevant coefficients, and narrowing the value ranges of certain cost drivers. We also develop an empirical formula for post-architecture estimate according to different types of abstract machines. Result shows that the revised model is capable of making reasonable estimates for projects using the B-Method.

As an empirical study, our approach needs to be tested and improved through further research on larger body of data and more kinds of projects. We are currently extending the approach for other formal methods including Z/Z++, VDM, SPECWARE [14], and PAR [18], which also support stated-based formal specifications and rewriting-based derivation and code generation as the B-Method.

5. ACKNOWLEDGMENTS

This work is supported in part by grants from National Grand Fundamental Research 973 Program (No. 2003CCA02800) of China. We also acknowledge the Academy of Armored Force Engineering for the assistance to our experimental projects.

6. REFERENCES


