Cost Estimation Driven Software Development Process

Orsolya Dobán, András Pataricza
Budapest University of Technology and Economics
Department of Measurement and Information Systems
Pázmány P. sétány 1/D IB.416, H-1117, Budapest, Hungary

E-mail: `[doban|pataric]@mit.bme.hu`

Abstract

The need for more and more dependable systems has been increased in the last decades. Strategic decisions during the design of these dependable systems require the joint control of the estimated cost and the product quality. Cost efficiency and development time became the most important factors of the software development process according to the international trends. The estimation of the cost remains a difficult problem till yet, despite of the use of standard, easy to understand modelling languages. On the other hand, no working environment is known supporting the gradually refined cost estimation derived automatically from the formal product models. In this paper the experiments of a pseudo cost estimation are detailed with a special emphasis on the cost estimation driven system design.

1. Introduction

A radical growth in software complexity was noticeable in the field of information technology in the former decades. These complex systems require the collaboration of large teams of programmers, where the scheduling and control of their activities are complex tasks.

Similarly, while traditional project management methodologies support the assessment of the feasibility of a software development plan in terms of time and human resources, only a minor fraction of methodologies support the estimation of the process related cost factors.

The current paper:

- gives a short overview on the main principles of the model-based cost estimation models, recapitulating the main features of one of the most widely used cost estimation model (COCOMO II);
- demonstrates the special features of using cost estimation in design flow aiming at dependable systems;
- summarizes the experiments of using COCOMO II in real software development projects;
- provides an overview on the weakness of the existing cost estimation methods;

2. Cost estimation

Cost estimates can and do occur at any point in the software development process. Already the most fundamental and rough cost estimator is useful from the very first phase on during the entire project life-cycle, as a decision support to the allocation of resources to a project.

A variety of cost estimation models was developed in the last two decades, including commercial, and public models as well.

All of these models consider several typical project factors like the size and complexity of the product, the human and the technology related characteristics of the environment, etc. On the basis of a set of real project time and effort log data they derive an own extrapolation based estimator of the cost and time of a project.

However some cost estimation models are not faithful enough to deliver proper predictions. The potential main origins of the errors in estimation are:

- oversimplified models (neglection of important factors),
- subjective parametrization of the model (the input parameters have a verbal definition only),
- statistically insufficient number or representativeness of the basic data set serving for extrapolation,
- difference between the state-of-the art technologies and the cost model.

2.1. Constructive cost model (COCOMO II)

A widely used model is the COCOMO II [3] among the existing model-based cost estimation methods, which has a complete public specification of its algorithms and interfaces.

The main formula of COCOMO II (Eq.1) expresses the predicted effort in the units of **Person Months (PM)**,

$$PM = A \cdot \left(\prod_{i=1}^{17} EM_i \right) \cdot Size^B \tag{1}$$

and it's inputs can be divided into three categories (Fig. 1):

- 5 scale-drivers, which are specific to the development process, and determine the value of the exponent B in the main COCOMO II formula
- 17 effort-multipliers (EM), related to the target software and to the development environment
- the estimated *Size* of the software to be developed in units of thousands of source lines of code (KSLOC). The goal is to measure the amount of intellectual work put into program development, but difficulties arise when trying to define consistent measures for different programming languages. Fortunately, additionally to the direct, heuristic estimation of the code length the function point (FP) based prediction can be used, as well. FP extrapolates the code size from the number and complexity of the product's designated functionality from the system requirement list.

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 1. Input parameters in the COCOMO II model

The large number of multipliers takes advantage of the greater knowledge available in the later development phases, to support gradually refined estimations. Each factor has an associated range of rating levels (”very low”, ”low”, ”nominal”, ”high”, ”very high”, ”extra high”). COCOMO II assigns to each qualitative category a corresponding numerical value. The first step of the cost estimation of a new project is the quantization of the factors into one of these categories.

2.1.1 Sensitivity

COCOMO II requires the estimation of altogether 22 parameter values. Similarly, as in the control theory, where partial derivates are used to characterize the sensitivity of the system to the changes in the model parameters, we use a similar approach to predict the consequences of a misjudgement in this subjective categorization process.

This way for instance, if we can assume, that categorization errors are confined to plus-minus a single category (e.g. a parameter has an assigned value of ”nominal” may have of the value between ”low” and ”high”) an uncertainty range can be estimated, as well. This way the designer has an uncertainty interval additionally to the expected value of the effort.

From the point of view of the sensitivity of the final cost estimator to these factors they can be classified into three categories:

- scale drivers
- effort multipliers, which have ascending function plotted against the increasing categories
- effort multipliers, which have descending function plotted against the increasing categories

1. To analyse a given **scale factor's** influence to the result (Fig.2) all the remaining input parameters were fixed, and the value of the analysed factor was changed between the categories of 'very low' and 'very high'. It can be seen that the higher the flexibility of the project, the lower the development cost is.

   Accordingly, the scale factor process maturity influences to the most, and the flexibility influences to the least important extent the result of the estimation. It means, that the miscategorization of flexibility will cause the smallest variance in the resulting value of the effort estimator.

2. **Ascending factors** are the product and the platform factors. It means, that increasing the value of these factors (e.g. the product's reliability) the development time will increase. The result of the sensitivity test (by fixed remaining input parameters) can be seen in Fig. 3, where only the two extreme functions are represented.

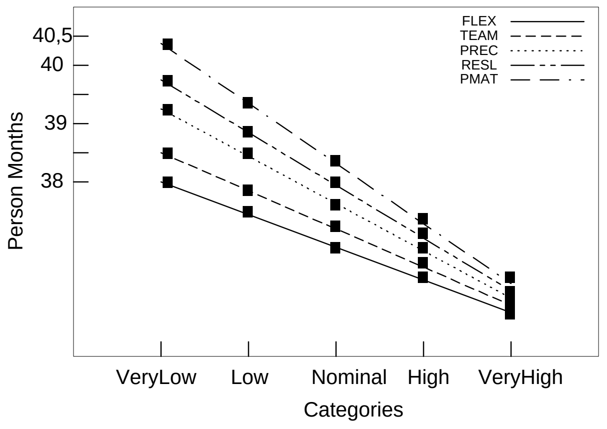

Figure 2. Sensitivity of the scale factors

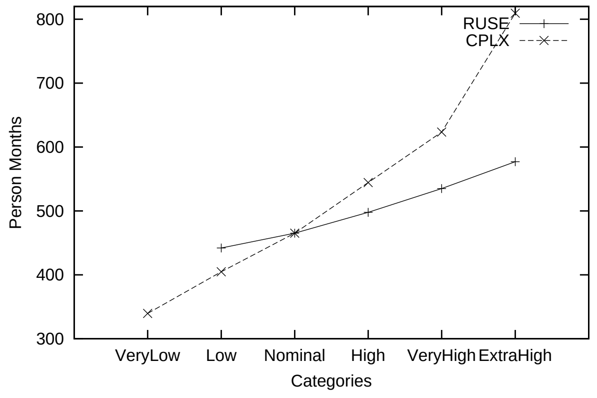

Figure 3. Ascending functions

By ascending functions the least sensible multiplier is the RUSE (Required Reusability), and the most sensible is the CPLX (Product Complexity). For instance fixing all the input parameters of the estimator on an arbitrary value, and changing only the category of the product's complexity from the 'very high' to the 'extra high' will increase the cost by 33%, while changing in the same way the reusability, the difference is only 10%.

3. The personnel and project factors are the **descending** ones. Increasing the category of these factors the estimated effort of the development process will decrease (Fig. 4).

   By descending functions the least sensible multiplier is the SITE (Multi-site Development), and the most sensible is the ACAP (Analyst Capability). Accordingly with training, and with improving the analysing capability of the developer team the cost of the development could be reduced to one of the highest degree.

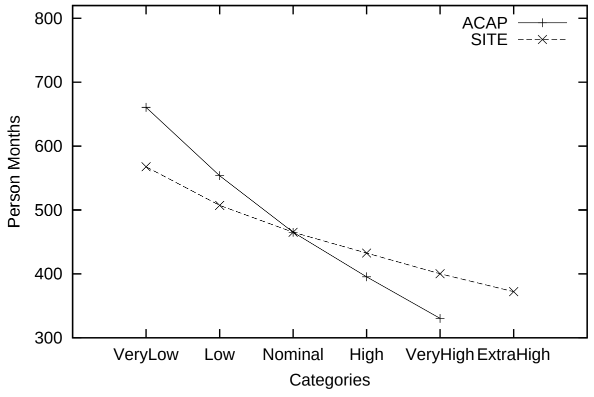

Figure 4. Descending functions

In this way all the input parameters can be represented like the function of the chosen category. On these curves the effect of the modification of a factor's value on the calculated cost can be observed, thus defining a range of uncertainty.

Accordingly the project manager has the possibility to decide about the composition of his developer team after accounting for the possible estimated effort.

3. System planning based on COCOMO II

COCOMO II estimates the cost and the effort of a software development process with the scalability of different parameters, like reliability, product complexity, execution time, etc.

In the following, we illustrate the usefulness of the cost estimation in a strategic decision process by a well-known simple example. A system of an intended high reliability is the objective of the software development process.

There are several ways for achieving high reliability of software systems by using different levels of redundancy. Two potential approaches are:

n-version programming: in which case n different solutions of the very same problem are elaborated by using strict design and implementation diversity. Their results will be compared by a single or by multiple distributed voter modules (Fig. 5).

recovery block (RB): after a checker module detects an error, it rejects the faulty results, and the calculations will be restarted by using an alternate implementation of the same problem (Fig. 6).

The main disadvantage of the recovery block principle is that in the case of fault, the repeated calculations are performed sequentially, thus increasing the computation time.

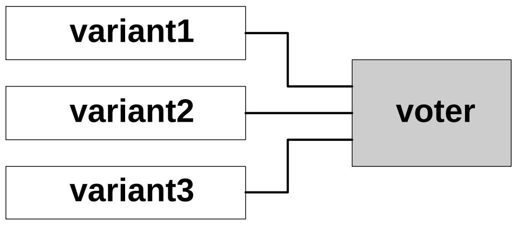

Figure 5. n-version programming

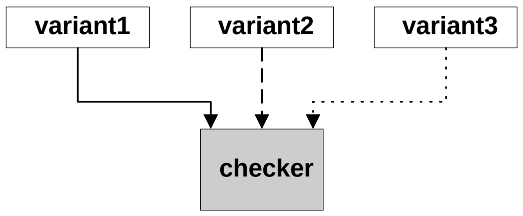

Figure 6. recovery block

For simplicity, we assume that no hard time limit exist for the applications, and both solutions are feasible.

Since both of the possibilities offer the same reliability for the entire system, the parameter of the decision can be the cost of the individual solutions.

3.1. Comparison of n-version programming and recovery blocks

Both the n-version and the RB will composed of four modules of the same size parameters, also three working variants (because of the needed redundancy), with 10.000 SLOC. Additionally the n-version will have a voter, and in turn the RB a checker module, both of them consisting of 2.000 SLOC.

Taking into account these peculiarities and conditions, the required development cost was predicted by the COCOMO II. By the calculation the input $Size$ parameter was determined by heuristic way, the effort multipliers, which differ from a value nominal, are summarized in the Figures 7 and 8.

For instance the product's reliability is less important by the different variants (that's why their value is nominal), however the voter module should have a very high reliability because of the decision about the correct output.

The n-version programming requires the strictly different development of the variants, including the programming language, the applied algorithms, etc. This way the developer team should have a more comprehensive knowledge, which is not fulfilled in the most cases. Therefore we suppose that both application and programming language experience are low during the development of the third module.

Substituting the predicted *Size* of the modules (10.000 SLOC/variants, 2.000 SLOC/voter and checker) and the quantitative values of the given factor categories into the main COCOMO formula, the result of the estimation is 156,2 PM for n-version programming, and 147,3 PM for the recovery block scheme, respectively.

No reference can be found in regard to the accuracy of the estimated efforts in the official COCOMO II manual [5]. We tested the approach by some benchmarks and by estimating some global measures for the uncertainty ranges in the form of sensitivity coefficients to the individual COCOMO model factors. The results of these experiments show, that the difference from the real effort is under 10%.

Summarizing this evaluation it coincides with the well-known fact, that although recovery blocks are slower (because of the sequential calculations in case of fault), this method is cheaper to develop. On the other hand the order of magnitude of the cost difference is an important aspect of the decision.

Although the size of the code was determined by heuristics, the ratio of the costs of several solutions is determinative.

4. Experiments of using COCOMO II

After analysing the model COCOMO II, the aim was to use it in real environment, through real software development projects in order to test its portability. There was the possibility to access the data of former software projects by a consulting company. Like any other consulting company this firm is specialized in the requirements analysis and the design of the target system, the implementation and the testing phase is not their task.

Effort log data were available only about the first two phases of the development. The partiality of information on the software development did not result in any difficulty, because the COCOMO II model provides results about the phases and subphases of the entire process, as well (Fig 9).

Five already finished projects were selected for the detailed analysis and post-estimation. By all projects the starting point of the experiment was the final version of the system's requirement list. It means, for determining the system's code size (input parameter of the estimator model) the **function point analysis** was chosen.

All the five projects were of the same order of complexity, which means all of them have had a very detailed function list in the same theme, in the database management.

The classification of the input parameters was the first arising problem, which was done by the participants of the given project. In the COCOMO II model manual [5] only verbal descriptions help the user to decide about the value of a factor, thus no exact criteria are given for controlling the unambiguous classification of these parameters.

EM	Var1	Var2	Var3	Voter	Comment
RELY	Nom	Nom	Nom	**VeryHigh**	Voter modules has to decide correctly
CPLX	Nom	Nom	Nom	**VeryHigh**	Voter module should synchronize its inputs, and count with the rounding errors
RUSE	Nom	Nom	Nom	**High**	General voter, used in other application as well
AEXP	Nom	Nom	**Low**	Nom	diff. implementation, diff. environment, diff. application experience
LTEX	Nom	Nom	**Low**	Nom	diff. implementation, diff. environment, diff. application experience

Figure 7. Effort multipliers for the n-version programming

EM	Var1	Var2	Var3	Checker	Comment
RELY	Nom	Nom	Nom	**VeryHigh**	Checker has to decide correctly
CPLX	Nom	Nom	Nom	**High**	The checker simply checks the feasibility of the result

Figure 8. Effort multipliers for the recovery blocks

For this reason the first main task was to do a **pseudo-estimation** for the first selected project, to examine which approach is relevant for the given company, like skill level of the developers, available development tools, etc.

After the first heuristical classification and calculation the result of the COCOMO II model was compared to the effort records. The result has shown 15% difference between the estimated and the real time effort.

The next task was to improve the factor quantization in order to reduce the difference between the estimation and the real costs. After covering near enough the required value the adjusted factor categories were stored in a database.

For the next four **post-estimations** these final parameter values were used as comparison point for determining the given project specific new factors. From this, the preparation for the estimation was much more easier, after determining the system's size the project team members only had to look for the differences between the stored and the new project characteristics.

According to the experiences there were parameters, which did not change through all the analysed projects. These factors are emphasized with a grey background in Fig. 10.

Basically these factors are the characteristics of either the developer team working permanently at the given company or the available tools, infrastructure and applied methodology during the project.

After identifying these fix, for a given team and environment continually valid parameters, the cost estimation becomes more simple.

4.1. Result of the post-estimations

The results of the post-estimations for the five analysed project are presented in Table 1.

Scale drivers	**Effort multipliers**
Precedentness	Required Software Reliability
Development Flexibility	Documentation ▨
Architecture/Risk Resolution ▨	Database Size
Team Cohesion ▨	Product Complexity
Process Maturity	Required Reusability
	Platform Volatility
	Execution Time Constraint
	Main Storage Constraint
	Personnel Continuity ▨
	Applications Experience
	Analyst Capability ▨
	Programmer Capability ▨
	Platform Experience ▨
	Language and Tool Exp. ▨
	Required Develop. Schedule
	Use of Software Tools ▨
	Multi-site Development

Figure 10. Fix input parameters

The pseudo-estimated project has the label "Proj1", consequently the difference from the real effort is only 2%. It's noticeable that in every case the resulted difference from the real data is under 10%.

4.2. Weakness of the experiment

Although the estimated results are extremely good, it shouldn't be forgotten that the experiments were done under favourable conditions, which have advantageously influenced the final result:

- All of the five projects were similar, both in the topic and real effort. For this reason after finishing the first pseudo-estimation the recalibration of the input parameters did not mean any difficulty.

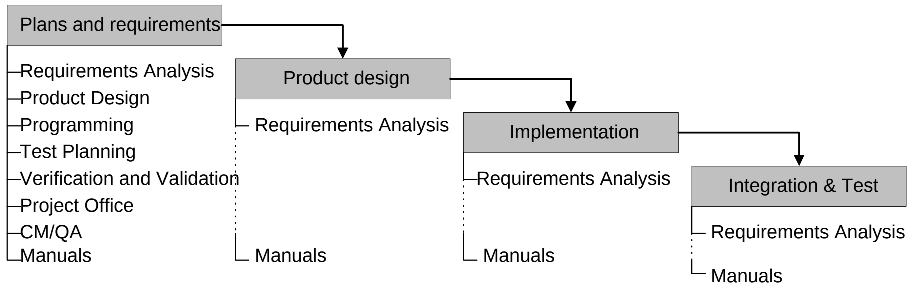

Figure 9. Software development phases by the COCOMO II model

	Proj1	Proj2	Proj3	Proj4	Proj5
SLOC (source lines of code)	5280	4000	5984	10464	7072
real effort in terms of time (hours)	186	159	392	411	282
estimated effort in terms of time (hours)	191	160	358	450	291
difference	**2%**	**1%**	**9%**	**9%**	**3%**

Table 1. Results of the post-estimations

- During the experiments only short-term projects were analysed, where the classified parameters are constant for the entire period of the development. However in long-term projects human factors gradually change due to the learning curve.

- Because of the post-estimation in all cases the final and complete version of the system requirement's list was used by the function point analysis, which did not contain the uncertainties characteristic for the early phases.

5. Conclusions

Modern post-prediction methods can help the designer in his technical decisions as well. In the field of dependability this methodology can be used to analyse solution alternatives and may play a central role in the current trend of constricting dependable systems from COTS elements by focusing effort to the crucial parts of the target system.

In the frame of an ongoing project our future aim is to fill the gaps in cost estimation, by integrating effort prediction with technical design activities. The designer will be supported in the selection of the optimal alternatives for implementation both in technical and economical turns.

References

[1] Murray R. Cantor: *Object-Oriented Project Management with UML*, John Wiley & Sons, Inc., 1998, Canada

[2] Ivar Jacobson:*Object-Oriented Software Engineering*, Addison-Wesly, 1992

[3] http://sunset.usc.edu/COCOMOII/

[4] http://sunset.usc.edu/research/COCOTS/index.html

[5] Center for Software Engineering: *COCOMO II Model Definition Manual*, 1998

[6] Dr. Barry Boehm: *Wirtschaftliche Software-Production*, Forkel-Verl., Wiesbaden, 1986

[7] B. W. Boehm, Ch. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. Reifer, B. Steece : *Software Cost Estimation With COCOMO II*, Prentice Hall, New Jersey, 2000

[8] A. Bondavalli, M. Dal Cin, D. Latella, and A. Pataricza: High-level integrated design environment for dependability (hide). In Proceedings of the IEEE Fifth International Workshop on Object-Oriented Real-time Dependable Systems (Words), IEEE Computer Society, 2000.

[9] ”Object-oriented modelling and optimization of industrial processes” (1999-2001, Foundation for the Hungarian Higher Education and Research)

[10] ”Automated Verification and Validation of UML-based Models of Information Systems” (1999-2001, Hungarian National Scientific Research Foundation)