Cost estimation driven software project management in UML

Orsolya Dobán, András Pataricza,
Budapest University of Technology and Economics,
Department of Measurement and Information Systems
e-mail: pataric@mit.bme.hu

Abstract

While in the former decades the design for dependability issue was confined to hard mission critical systems, the widespread use of online services raises the same question in an increasingly wide spectrum of applications. However, there exist basic differences between the design of mission critical applications and on-line services. Mission critical applications can tolerate a high degree of redundancy and the related cost overhead compared to a pure functional solution. However everyday's applications require only a high availability of the applications as an add-on to the quality of the system, but do not allow for a significant overhead.

1 Introduction

Today's programming challenges call for the collaboration of large teams of programmers to develop large complex programs. The scheduling and control of the work of development teams are complex tasks. No single working environment is known, covering all phases of the software development process.

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

A frequently asked question, that how to estimate the cost and effort of a project, and how reliable are these estimations. Whether this cost estimation is extendable, and if yes, than for what else is it usable.

The current paper:

- gives an overview on the usability of semi-formal models in the formulation of software development projects by using a special dialect of the Unified Modeling Language (UML)
- recapitulates the most widely used cost estimation model COCOMO II, with a special emphasis on designing dependable systems
- provides an overview on the relationship of UML and COCOMO II (how to use the cost estimators for system planning, how to automate the cost estimation during the system design in the same environment).

2 Project management and UML

Both the planning and execution of a development project require an intensive communication and collaboration between the members of the development team. This way, the management of a project and the development of a software require the supporting diagrams and documents. This diagrams have to use a common graphic language accepted by every member of the project team in order to support mutual understanding.

But which diagrams should be created, what time, and by whom? There are many crucial open questions in a project, which can cause problems, like:

1. *Inadequate and unstable requirements:*

   There are two common causes for insufficient requirement specification:

   - the customer often does not fully understand his own requirements
   - even if the customer is capable of thinking through the well founded specification of the complex functionalities required of the system under development, he may have difficulties in properly documenting this requirements.

   One of the most successful solutions for capturing user requirements is the use case formalism, [2] a method for specifying and managing dynamic requirements : "A **use case** is a coherent unit of functionality provided by a system or class as manifested by sequences of messages exchanged among the system and one or more outside inter-actors (called actors) together with actions performed by the system."

   Please note, that the use case diagrams are able to capture graphically the dependability requirements of a system as well, in a similar fashion as the functional specification. For instance, the formalism allows for expressing which services are the most important ones to be kept alive in the case of a degradation, or what are the fallback solutions in the case of a system failure.

   This way, it is crucial that a continuous monitoring of costs and reliability should become an integral part of the project management process. In the current paper we present a new method of combining the de facto standard notation of UML with one of the best cost estimates methodologies known, COCOMO II and show its usefulness in the application of designing dependable systems.

2. *Poor team communications:*

   Clearly, communication among team members is imperative. This communication with and within the team needs a common language. The project manager, therefore, must establish a framework for precise communication. The implementation of a standard object modeling language is one component of this communications framework.

   UML is on its way to being adopted by the industry as the standard, general-purpose visual modeling language designed to specify, visualize, construct and document the artifacts of a software system. [3] The UML provides a set of static and dynamic diagrams, such as package diagrams.

3. *Inadequate customer communications:*

   To avoid this problem it is necessary to create a sufficiently detailed documentation, in order to cover all important aspects of the functional and dependability specification and the implementation.

   A potential disadvantage is that if the developers produce documentation only in the very final phase of the software development, the result is frequently an incomplete documentation. In this way modern methodologies aim at a self documentation feature which assures the consistency between the software and the documentation.

4. *Unnecessary complexity:*

   A program with a large number of state transitions and numerous code paths is complex. A high degree of program complexity can drastically reduce the testability of the program, this way latent errors can remain even after this competition of the software development process. Latent errors can lead to a drastically reduced quality of services for the end user.

   To overcame this problems it is necessary to encapsulate the work of each individual programmer in a similar fashion, for instance as the software modules.

5. *Ineffective team behavior:*

   Programmers take pride in their individual accomplishments and their ability to solve difficult problems, and they are proud that they can bend the complicated computer platforms to their individual will. [1] However UML can be applied not only to model the target product, but to model the development process leading to it, as well. *So the solutions are: creating use case diagrams, package*

diagrams, generating documentation, and applying object-oriented methods. All, what is needed, is present in UML! Only the different diagrams have to change their interpretation. Not only the modeling of the target product, but the modeling of the development process is a candidate aim.

3 Quantitative analysis and optimization

Cost estimates can and do occur at any point in the software development process. Already the most fundamental cost estimator is useful in the very first phase of the project life-cycle. A company must decide on the commitment of resources to a project, based on an estimate of its resources requirements.

There are several methods for creating estimates, approximating the cost, effort, and schedule when planning a new software development activity. [4]

3.1 Constructive Cost Model (COCOMO)

From this point on, we refer by COCOMO both the original method [5] and its refinement known as COCOMO II. COCOMO II's relationships, algorithms and interfaces are available for the wide public.

The COCOMO II model is the result of several serious examinations of many projects. USC COCOMO II.1998.0 beta has been calibrated to 161 data points, (cost traces of software development projects), to blending empirical data with expert opinion to calibrate the model parameters.

Hereafter we will focus only to those features of COCOMO II, which are relevant to the design of dependable systems.

3.1.1 Theoretical bases

In the base model of COCOMO effort is expressed in the units of **Person Months (PM)**, amount of time one person spends working on the software development project for one month. (Eq. 1)

$$PM = A \cdot Size^B \tag{1}$$

The inputs are the estimated *Size* of the software to be developed in units of thousands of source lines of code (KSLOC), and two constants, A, and the scale factor, B. Here, the magnitude of the effort is a linear function of the size of the project.

COCOMO II. is a refined version of the original COCOMO, it takes more factors of a process into account, as in the original COCOMO. (Eq. 2.)

$$PM = \prod_{i=1}^{17}(EM_i) \cdot A \cdot \left[\left(1 + \frac{BRAK}{100}\right) \cdot SIZE\right]^B \tag{2}$$

In COCOMO II. the value of B depends on the values of five factors, the so-called scale drivers. (Table 1) There are 17 effort multipliers (EM). The large number of multipliers takes advantage of the greater knowledge available in the later development phases.

Each factor has a range of rating levels, from "Very Low" to "Extra High". Each rating level has an associated numerical weight. For instance the nominal weight for an Effort Multiplier is 1.0. If a rating level causes more software development effort, then its corresponding EM weight is above 1.0.

4 System planning based on COCOMO

COCOMO II supports the estimation of the cost and the effort of a software development so, that the different parameters, like reliability, product complexity, execution time, etc. are scalable. Because of these facts, this cost estimation is extremely useful for system planning.

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

Scale drivers	**Effort Multipliers**	
Precedentness	*Product Factors:*	Required Software Reliability(RELY)
Development Flexibility		Documentation
Architecture/Risk Resolution		Database Size
Team Cohesion		Product Complexity(CPLX)
Process Maturity		Required Reusability(RUSE)
	Platform Factors:	Platform Volatility
		Execution Time Constraint
		Main Storage Constraint
	Personnel Factors:	Personnel Continuity
		Applications Experience(AEXP)
		Analyst Capability
		Programmer Capability
		Platform Experience
		Language and Tool Experience(LTEX)
	Project Factors:	Required Development Schedule
		Use of Software Tools
		Multi-site Development

Table 1: Scale drivers (B) and Effort Multipliers (EM)

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

1. **N-Version Programming:**
   in which case N different versions of the very same problems are elaborated by using strict design and implementation diversity. Their results will be compared by a single or by multiple and distributed, so called voter modules.

2. **Recovery Block:**
   When a checker module with a high probability of error detection detects an error, it rejects the faulty results, and the calculations will be restarted by using an other alternate implementation of the same problem.

However the main disadvantage of the Recovery Block is, that in the case of a fault, the repeated calculations are performed sequentially, thus increasing the computation time.

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

In system planning for comparing different possible configurations of the target system, which conformations include different number and types of modules, is needed the cost countability separately for the individual modules too. The COCOMO II is useful for determining the cost of individual modules, and comparing which modules are accounting for the highest part of the cost. [6]

4.1 Comparison of N-Version Programming and Recovery Block

If both of the chosen solutions with the given parameters (number of modules (n), reliability of the individual modules) result in the same reliability of the entire system,the parameter of the decision can be the cost of the individual solutions.

Both the N-Version and the RB would include four modules with the same size parameters, also three working variants (because of the needed redundancy), and additionally the N-Version would have a voter module, the RB in turn a checker module.

Only the Effort Multipliers, which differ from a value nominal(1.0), are summarized in the Tables 2 and 3.

EM	Var1	Var2	Var3	Voter	comment
RELY	Nom	Nom	Nom	**VeryHigh**	Voter module 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, for using in other applications too.
AEXP	Nom	Nom	**Low**	Nom	diff. implementations, in diff. environments, diff. experience
LTEX	Nom	Nom	**Low**	Nom	diff. implementations, in diff. environments, diff. experience

Table 2: 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 results

Table 3: Effort Multipliers for the Recovery Blocks

The results of the effort estimation:

N-Version Programming: $Effort_{N-Version} = 156,2 PersonMonths$

Recovery Block: $Effort_{RecoveryBlock} = 147,3 PersonMonths$

The result of this evaluation shows, that although Recovery-Blocks are slower (because of the sequential calculations in case of fault), this method is cheaper to develop.

5 COCOMO II and the UML

Our goal was a combination of COCOMO II with the UML based design process in order to utilize the target design related information embedded into its UML model for providing proper cost estimates.

Please note, that as the hierarchical model refinement process reveals more and more details on the target design, the inclusion of the characteristics automatically derived from them will gradually result in an increasingly accurate cost estimate.

5.1 The advantages of UML-COCOMO combination

1. In software development project management needs the description of the available resources. It means, that the available platforms, personnel capacities, technological tools are to be described in UML, because these are the hierarchical elements of the development-environment.

   For instance a developer can be the part of a static class diagram. He is classifiable into the different COCOMO II classes by the personnel factors. The values of this attributes (like application experience, platform experience, and so on) can be determined for every person, like the attributes of the given class by an enumeration type.

   Accordingly the requirement-system of COCOMO is describable by a static class diagram in UML

2. The COCOMO II can be jointly used with UML not only for system planning in the first phase, but also during the subsequent design phases, it is possible to execute new estimations. The gradually refined diagrams can contain the COCOMO parameters too, so as the design flow advances the same method, as described above, supports automatically more and more realistic cost estimations.

If this refinement process traverses systematically the system, the elementary subcomponents will be reached at the very end. Because of the refining, for these elementary applications it is possible to get a narrow cost estimation.

Accordingly to every application the value of its cost estimation can belong as an additional parameter.

3. The system cost can be estimated by totalizing the costs of the individual components. Please note, that this composite estimator takes the component integration into account, as well instead of simple summing up the partial costs.

   Accordingly if someone would like to perform a cost estimation for a system, he does not need an exact knowledge on the implementation of all its parts. If he has a database, which includes a proper cost estimator of the different applications, than the system cost estimator can be derived.

   Moreover, a part of the skill for proper cost estimation can be hidden in stereotype libraries.

   Accordingly it is possible to execute a cost estimation helped by stereotypes. It is not a startler, because the COCOMO is the generalization of an experience too.

   This way UML library can function as a knowledge base.

5.2 The realization of the UML-COCOMO combination

The now existing method for creating automatic estimations with the COCOMO II model in the UML environment is the following one:

By using the CASE tool Innovator, it is possible to write comments to every part of the designed system, and to generate documentation automatically with these comments (including the needed parameters for the COCOMO). With a script - using the generated file - it is very simple to produce the results of the estimation.

6 Summary

A method was shown, which estimates cost and effort by using the scalable property of different parameters in the cost estimation model, COCOMO II.

The Unified Modeling Language is useful for software development project management, and it is expedient to integrate this UML model with the COCOMO II. That rate it is possible to create estimations automatically during the design phases, and to generate all the system characteristics by model transformation.

References

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

[2] Ivar Jacobson: Object-Oriented Software Engineering

[3] UML documentation: http://www.rational.com/uml/index.jtmpl

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

[5] Dr. Barry Boehm: Software Engineering Economics, 1981

[6] COCOMO II Model Definition Manual (downloaded from http://sunset.usc.edu/COCOMOII/cocomo.html)

[7] Boehm, Barry W.: Wirtschaftliche Software-Production, Forkel-Verl., Wiesbaden, 1986