



OO-COCOMO Estimation Technique

Version 1.0

Jim Ladd
Wazee Group, Inc.

November 15, 2000



Contents

Introduction.....	3
Background.....	3
Approach.....	3
In Practice	6
Moving Forward	7
References.....	7

Introduction

The paper presents the fusion of the Constructive Cost Model (COCOMO) with one of the first published estimation techniques for object-oriented software. The OO-COCOMO technique allows the user to estimate the effort required to construct a software application based on the number of business objects.

Background

In 1981, Barry Boehm published the *Software Engineering Economics* book, a milestone in the software development timeline. The cornerstone of this work is the Constructive Cost Model (COCOMO) which facilitates the estimation of the effort, cost, and schedule of software development projects. The model in the most basic form requires only two inputs; 1) the size of the project in number of thousands of lines of code, and 2) the type of project. With these two inputs, the effort (staff-months) can be calculated along with the development time (months), and the average staff size (staff/month).

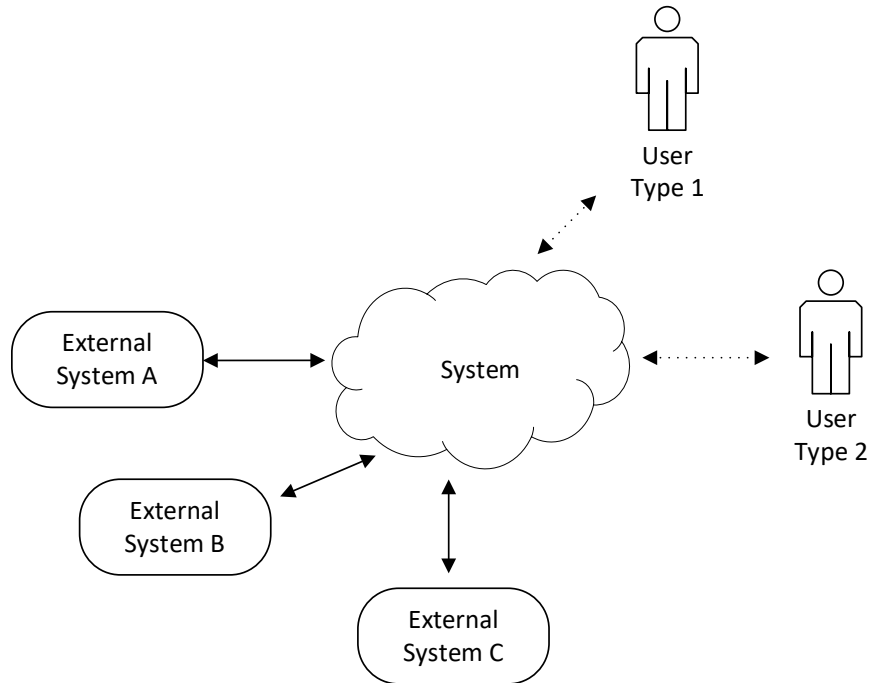
The challenge with the COCOMO technique is the determination of the project size in thousands of lines of code (KLOC). One of the most common approaches for project size estimation is the Work Breakdown Structure (WBS). This technique decomposes the solution domain into smaller, hierarchical components until the size of the unit of work can be estimated with a high degree of confidence. This process is usually very tedious and difficult in order to achieve an overall accurate estimate.

In the mid-1980s, the software development universe shifted. Object-oriented programming was rapidly gaining popularity. Not only were OO languages being embraced, the object-oriented paradigm also impacted the analysis, design, and testing phases of the software development lifecycle. One of the last areas to be invited to the party was software project estimation. It wasn't until 1994 that the first book (*Object-Oriented Software Metrics* by Mark Lorenz and Jeff Kidd) on estimation techniques for OO projects was published.

The estimating process presented in the *Object-Oriented Software Metrics* book begins with the identification of the number of key objects for the domain of interest. The output of the process is a number of lines of code for the project. After acquiring the book in 1994, I decided to merge this estimation approach with the Constructive Cost Model to create the OO-COCOMO technique.

Approach

The first step of OO-COCOMO technique is to define the boundary of the problem space. This activity identifies the entities that are external to the project and the subject matter that lies within. These entities include the types of users and the external systems that will be interacting with the system.



After the boundary is set, the system (i.e. the problem space) can be analyzed and partitioned into core domains and common services. The following estimation steps are applied to each domain that requires custom code.

1. Identify the key objects within the domain.

The key objects can be extracted from scenarios derived from the use cases, review of the interface documents from the external systems, storyboards from the user interfaces, etc.

2. Categorize the type of user interface for the domain.

From the *Object-Oriented Software Metrics*, the types of user interfaces and the multipliers are listed below:

User Interface Type	Multiplier
No UI	2.0
Simple, text-based	2.25
Graphical	2.5
Complex, drag-and-drop	3.0

3. Determine the number of support classes

Multiply the number of key objects with the UI Type Multiplier to determine the number of objects.

4. Determine the number of public methods

Multiple the number of objects with the average number of public methods per object. This number can be derived from existing codebases within the company. In the *Object-Oriented Software Metrics* book, the authors use an average of 20 for the C++ language.

5. Determine the number of lines of code

Multiple the number of public methods by the number of lines of code per method. This number, like in the previous step, can be derived from existing codebases within the company. In the *Object-Oriented Software Metrics* book, the authors use an average of 24 for the C++ language.

6. Calculate the total number of lines of code

Sum the number of lines of code for each domain to obtain a total KLOC value.

7. Classify the overall project mode

Select one of the project modes. The associated *a*, *b*, *c*, and *d* values will be used in the following equations.

Project Mode	Description	a	b	c	d
Organic	Small teams, familiar software, flexible environment	2.4	1.05	2.5	0.38
Semi-Detached	Medium teams, mixed experience, moderate constraints	3.0	1.12	2.5	0.35
Embedded	Large teams, tight hardware/real-time constraints, high complexity	3.6	1.20	2.5	0.32

8. Calculate the effort in staff-months

The total effort (E) is measured in staff-months and is calculated with the following equation:

$$E = a * (KLOC)^b$$

9. Calculate the time for development

The total time (T) is measured in months and is calculated with the following equation:

$$T = c * (E)^d$$

10. Calculate the average staff size

The average number of staff members required over the project schedule is the total effort divided by the time for development:

$$\text{Average Staff Size} = E / T$$

The output of OO-COCOMO is the level of effort (E) in staff-months, the time for development (T) in months, and the average staff size. For more sophisticated forecasting, the COCOMO model offers a set of modifiers that will impact the effort calculation. These modifiers allow a more accurate estimate by capturing more granular classifications of the specific project.

The default value of each factor is 1.0 which does not impact the baseline effort. The product of the multipliers is multiplied to the effort (E) to obtain the final level of effort.

Project Factor	Key	Very Low	Low	Nominal	High	Very High	Extra High
Reliability	RELY	0.75	0.88	1	1.15	1.4	
Data base size	DATA		0.94	1	1.08	1.16	
Product complexity	CPLX	0.7	0.85	1	1.15	1.3	1.65
Execution time constraint	TIME			1	1.11	1.3	1.66
Main storage constraint	STOR			1	1.06	1.21	1.56
Virtual machine volatility	VIRT		0.87	1	1.15	1.3	
Computer turnaround time	TURN		0.87	1	1.07	1.15	
Analyst capability	ACAP	1.46	1.19	1	0.86	0.71	
Applications experience	AEXP	1.29	1.13	1	0.91	0.82	
Programmer capability	PCAP	1.42	1.17	1	0.86	0.7	
Virtual machine experience	VEXP	1.21	1.1	1	0.9		
Programming language experience	LEXP	1.14	1.07	1	0.95		
Modern programming practices	MODP	1.24	1.1	1	0.91	0.82	
Software tools	TOOL	1.24	1.1	1	0.91	0.83	
Required development schedule	SCED	1.23	1.08	1	1.04	1.1	

In Practice

Six years ago, I combined my favorite traits from the *Software Engineering Economics* and the *Object-Oriented Software Metrics* books into the OO-COCOMO technique. To facilitate the estimation process, I created a simple spreadsheet that allows the easy entry of the inputs and quick calculations of outputs.

I first used this technique with in a large health care company on several embedded, hard real-time projects. Each estimate proved to be close to the manual WBS-based forecasts and required less time and effort to generate.

I also used this technique over a two-year period at a global IT consulting company. The company's common practice was to send 2-3 consultants to a client for a period of 2 weeks in order to generate schedule and cost estimates for a new software project. At the end of this period, one of the consultants would reach out and request a "confirmation" estimate. After a few questions, the OO-COCOMO calculation was always within a 5% range of the manual estimate.

Moving Forward

While the OO-COCOMO technique has been extremely useful in the past, the future demands constant evolution and refinement. Technologies like open-source frameworks and Computer Aided Software Engineering (CASE) tools promise smaller custom codebases. Fewer KLOCs of custom work should result in greater accuracy with the estimation. Iterative development process models like Agile will require an initial estimate and then "in-flight" updates of the estimate. The application of design patterns have also increased productivity of engineers and should be reflected in the estimation mode.

"To improve is to change; to be perfect is to change often." — Winston S. Churchill

Truer words have not been written about software development.

References

Boehm, B. W. (1981). *Software engineering economics*. Prentice Hall

Lorenz, M. and Kidd, J. (1994). *Object-oriented software metrics*. Prentice Hall