Cost Estimation Models

Most of the work in the cost estimation field has focued on algorithmic cost modelling. In this process costs are analysed using mathematical formulas linking costs or inputs with metrics to produce an estimated output. The formulae used in a formal model arise from the analysis of historical data. The accuracy of the model can be improved by calibrating the model to your specific development environment, which basically involves adjusting the weightings of the metrics.
Generally there is a great inconsistency of estimates. Kemerer [Kemerer 93] conducted a study indicating that estimates varied from as much as 85 - 610 % between predicated and actual values. Calibration of the model can improve these figures, However, models still produce errors of 50-100%.

- **SLIM (Software Life Cycle Management)**

Putnam's [Putnam 78] SLIM is one of the first algorithmic cost model. It is based on the Norden/Rayleigh function and generaly known as a macro estimation model (It is for large projects). SLIM enables a software cost estimator to perform the following functions :

- **Calibration** Fine tuning the model to represent the local software development environment by interpreting a historical database of past projects.
- **Build** an information model of the software system, collecting software characteristics, personal attributes, computer attributes etc.
- **Software sizing** SLIM uses an automated version of the lines of code (LOC) costing technique.

The algorithm used is :

$K = (LOC / (C * t^{4/3})) * 3$

K is the total life cycle effort in working years, *t* is development and the *C* is the technology constant, combining the effect of using tools, languages, methodology and quality assurance (QA). time in years.
The value of technology constant varies from 610 to 57314. For easy, xperienced projects technology constant is high.

SLIM is not widespreadly used but there is a SLIM tool.

Advantages of SLIM

i. Uses linear programming to consider development constraints on both cost and effort.
ii. SLIM has fewer parameters needed to generate an estimate over COCOMO'81 and COCOMO'II

Drawbacks of SLIM

i. Estimates are extremely sensitive to the technology factor
ii. Not suitable for small projects

- **COCOMO'81 (Constructive Cost Model)**

Boehm's [Boehm 81] COCOMO model is one of the mostly used model commercially. The first version of the model delivered in 1981 and COCOMO II is available now.

COCOMO'81 is derived from the analysis of 63 software projects in 1981. Boehm proposed three levels of the model : **Basic, intermediate, detailed.**

- **The basic COCOMO'81 model** is a single-valued, static model that computes software development effort (and cost) as a function of program size expressed in estimated lines of code (LOC).
- **The intermediate COCOMO'81 model** computes software development effort as a function of program size and a set of "cost drivers" that include subjective assessments of product, hardware, personnel, and project attributes.
- **The detailed COCOMO'81 model** incorporates all characteristics of the intermediate version with an assessment of the cost driver's impact on each step (analysis, design, etc.) of the software engineering process.

COCOMO'81 models depends on the two main equations :

First is *development effort* (based on MM - man-month / Person-month / staff-month is one month of effort by one person. In COCOMO'81, there are 152 hours per Person-Month. According to organization this values may differ from the standard by 10% to 20%.) for **the basic model** :

$MM = aKDSI^b$

Second is *effort and development time* (TDEV)

$TDEV = cMM^d$

KDSI means the number of thousand delivered source instructions and it is a measure of **size**
The coefficents a, b, c and d are depent on the *mode of the development.* There are three modes of development :

Development Mode	**Project Characteristics**			
	Size	*Innovation*	*Deadline/constraints*	*Dev. Environment*
Organic	Small	Little	Not tight	Stable
Semi-detached	Medium	Medium	Medium	Medium
Embedded	Large	Greater	Tight	Complex hardware/customer interfaces

Here is the coefficents related to development modes for **intermediate model** :

Development Mode	**a**	**b**	**c**	**d**
Organic	3.2	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Basic mode uses only **size** in estimation. Intermadiate mode also uses 15 *cost drivers* as well as size.

In **intermediate mode** development effort equation becomes :

$MM = aKDSI^b C$

C (effort adjustment factor) is calculated simply multiplying the values of cost drivers. So the intermediate model is more accurate than the basic model.
The steps in producing an estimate using the intermediate model COCOMO'81 are :

1. Identify the mode (organic, semi-detacted, embedded) of development for the new product.
2. Estimate the size of the project in KDSI to derive a nominal effort prediction.
3. Adjust 15 cost drivers to reflect your project.
4. Calculate the predicted project effort using first equation and the effort adjustment factor (**C**)
5. Calculate the project duration using second equation.

Example estimate using the intermediate COCOMO'81

Mode is *organic*
Size = 200KDSI
Cost drivers :

- Low reliability => .88
- High product complexity => 1.15
- Low application experience => 1.13
- High programming language experience => .95
- Other cost drivers assumed to be nominal => 1.00

$C = .88 * 1.15 * 1.13 * .95 = 1.086$

$Effort = 3.2 * (200^{1.05}) * 1.086 = 906 MM$

$Development time = 2.5 * 906^{0.38}$

Advantages of COCOMO'81

i. COCOMO is transparent, you can see how it works unlike other models such as SLIM.
ii. Drivers are particularly helpful to the estimator to understand the impact of different factors that affect project costs.

Drawbacks of COCOMO'81

i. It is hard to accurately estimate KDSI early on in the project, when most effort estimates are required.
ii. KDSI, actually, is not a size measure it is a length measure.
iii. Extremely vulnerable to mis-classification of the development mode
iv. Success depends largely on tuning the model to the needs of the organization, using historical data which is not always available

---

- **COCOMO'II (Constructive Cost Model)**

Researchs on COCOMO II is started in late 90s because COCOMO'81 is notenough to apply to newer software development practices. You can find latest information about COCOMO'II from COCOMO II Home Page

COCOMO'II differs from COCOMO'81 with such differences :

- COCOMO'81 requires software size in KSLOC as an input, but COCOMO'II provides different effort estimating models based on the stage of development of the project.
- COCOMO'81 provides point estimates of effort and schedule, but COCOMO'II provides likely ranges of estimates that represent one standard deviation around the most likely estimate.
- COCOMO'II adjusts for software reuse and reengineering where automated tools are used for translation of existing software, but COCOMO'81 made little accommodation for these factors
- COCOMO'II accounts for requirements volatility in its estimates.
- The exponent on size in the effort equations in COCOMO'81 varies with the development mode. COCOMO'II uses five scale factors to generalize and replace the effects of the development mode.

COCOMO II has three different models :

- The Application Composition Model

  Suitable for projects built with modern GUI-builder tools. Based on Object Points.

- The Early Design Model

  This model is used to make rough estimates of a project's cost and duration before it is entire architecture is not determined. It uses a small set of new Cost Drivers, and new estimating equations. Based on Unadjusted Function Points or KSLOC.

  For the Early Design and Post Architecture Model :

$$Effort = \frac{ASLOC\left(\frac{AT}{100}\right)}{ATPROD} + a\left[size\left(1+\frac{BRAK}{100}\right)\right]^b \prod EM_i$$

$$b = 1.01 + \frac{1}{100}\sum SF_j$$

$$Size = KSLOC + KASLOC\left(\frac{100-AT}{100}\right) AAM$$

  Where a = 2.5, SF_j = scale factor, EM_i = effort multiplier
  BRAK = Percentage code discarted due to requirement volatility
  ASLOC = size of adapted components
  AT = percents of components adapted
  ATPROD = Automatic Translation Productivity
  AAM = Adaptation Adjustment Multiplier

  COCOMO'II adjusts for the effects of reengineering in its effort estimate. When a project includes automatic translation, following list must be estimated :

  - Automatic translation productivity (ATPROD), estimated from previous development efforts
  - The size, in thousands of Source Lines of Code, of untranslated code (KSLOC) and of code to be translated (KASLOC) under this project.
  - The percentage of components being developed from reengineered software (ADAPT)
  - The percentage of components that are being automatically translated (AT).

  The effort equation is adjusted by 15 cost driver attributes in COCOMO'81, but COCOMO'II defines seven cost drivers (EM) for the Early Design estimate:

  - Personnel capability
  - Product reliability and complexity
  - Required reuse
  - Platform difficulty
  - Personnel experience
  - Facilities
  - Schedule constraints.

  Some of these effort multipliers are disaggregated into several multipliers in the Post-Architecture COCOMO'II model.

  COCOMO'II models software projects as exhibiting decreasing returns to scale. Decreasing returns are reflected in the effort equation by an exponent for SLOC greater than unity. This exponent varies among the three COCOMO'81 development modes (organic, semidetached, and embedded). COCOMO'II does not explicitly partition projects by development modes. Instead the power to which the size estimate is raised is determined by five scale factors:

  - Precedentedness (how novel the project is for the organization)
  - Development flexibility
  - Architecture/risk resolution
  - Team cohesion
  - Organization process maturity.

- The Post-Architecture Model

  This is the most detailed COCOMO II model. It is used after project's overall architecture is developed. It has new cost drivers, new line counting rules, and new equations.

  Use of reengineered and automatically translated software is accounted for as in the Early Design equation (ASLOC, AT, ATPROD, and AAM). Breakage (BRAK), or the percentage of code thrown away due to requirements change is accounted for in 2.0. Reused software (RUF) is accounted for in the effort equation by adjusting the size by the adaptation adjustment multiplier (AAM). This multiplier is calculated from estimates of the percent of the design modified (DM), percent of the code modified (CM), integration effort modification (IM), software understanding (SU), and assessment and assimilation (AA). Seventeen effort multipliers are defined for the Post-Architecture model grouped into four categories:
  - Product factors
  - Platform factors
  - Personnel factors
  - Project factors

  These four categories parallel the four categories of COCOMO'81 - product attributes, computer attributes, personnel attributes and project attributes, respectively. Many of the seventeen factors of COCOMO'II are similar to the fifteen factors of COCOMO'81. The new factors introduced in COCOMO'II include required reusability, platform experience, language and tool experience, personnel continuity and turnover, and a factor for multi-site development. Computer turnaround time, the use of modern programming practices, virtual machine experience, and programming language experience, which were effort multipliers in COCOMO'81, are removed in COCOMO'II.

  A single development schedule estimate is defined for all three COCOMO'II models :

$$Schedule = c\left[Effort^{0.33+0.2(b-1.01)}\right]\frac{SCED\%}{100}$$

$$b = 1.01 + \frac{1}{100}\sum SF_j$$

  Where c = 3, SF_j scale factor and SCED% = schedule compression/expansion parameter.

Sencer's Home Software Measurement Page

sencer@hun.edu.tr
This page is maintained by Ümit Karakaş and Sencer Sultanoğlu
Last modified Oct 12, 98