COCOMO
(Constructive Cost Model)

Seminar on Software Cost Estimation
WS 2002 / 2003

presented by
Nancy Merlo – Schett

Requirements Engineering Research Group
Department of Computer Science
University of Zurich, Switzerland

Prof. Dr. Martin Glinz
Arun Mukhija

I. CONTENT

II. ABSTRACT

The original COCOMO stands for *Constructive Cost Model*. The word "constructive" implies that the complexity of the model can be understood because of the openness of the model, which permits exactly to know WHY the model gives the estimates it does. The model was first published by Dr. Barry Boehm in 1981, and reflected the software development practices of these days. Since this time many efforts were done in the improvement of the software development techniques. Some of the changes were moving away from mainframe overnight batch processing to real time applications, strenuousness in effort in building software for reusing, new kind of system development in including off-the-shelf software components (COTS) and spending as much effort on designing and managing the software development process as was once spent creating the software product.

These changes urged to revise the existing model. By the joint efforts of USC-CSE (University of California, Center for Software Engineering) and the COCOMO II Project Affiliate Organizations the COCOMO II model was presented, which should remedy all deficiencies. This new, improved COCOMO (COCOMO II) is now ready to assist professional software cost estimators.

III. AFFILIATES

The work for COCOMO II has been supported financially and technically by the COCOMO II Program Affiliates: Aerospace, Air Force Cost Analysis Agency, Allied Signal, DARPA, DISA, Draper Lab, EDS, E-Systems, FAA, Fidelity, GDE Systems, Hughes, IDA, IBM, JPL, Litton, Lockheed Martin, Loral, Lucent, MCC, MDAC, Microsoft, Motorola, Northrop Grumman, ONR, Rational, Raytheon, Rockwell, SAIC, SEI, SPC, Sun, TASC, Teledyne, TI, TRW, USAF Rome Lab, US Army Research Labs, US Army TACOM, Telcordia, and Xerox.

IV. OVERVIEW

The paper gives an overview of COCOMO. It is structured in two parts.

Part 1 (section 1) introduces COCOMO I or COCOMO '81 from Dr. Barry Boehm. This part is hold very short. It summarizes only the most important issues. Because of the up-to-dateness more attention is given to part 2 (see below). COCOMO I provides a well-defined, open engineering basis for reasoning about the cost and schedule implications of a software solution. Three levels are discussed and a conclusion shows the advantages and drawbacks of the model. The detailed information about ratings and cost drivers can be found in [Boehm 81].

Part 2 (section 2 to 10) deals with COCOMO II. At the beginning an overall context is given. The Model Definition then presents the specific definitions of COCOMO II. The three-stage model is introduced followed by the explanation of its quantities, estimating equations, scale factors, cost drivers and rating scales. The rating scales themselves won't be discussed in this paper.

The section about the Bayesian approach gives a brief overview about this calibration process and summarizes its results.

The emerging extensions are described in section 6. At the beginning of this section a status quo summarizes the current state of these extensions as most of them are still experimental (the calibration and counting rules aren't yet robust).

The outlook, some general thoughts about the problems with existing models, a critic and a list of available tools close this part.

At the end of this paper the list of references is given.

CHARTER1: COCOMO I, COCOMO'81

1 COCOMO I

In this section the model of COCOMO I also called COCOMO'81 is presented. The underlying software lifecyle is a waterfall lifecycle. Detailed information about the ratings as well as the cost drivers can be found in [Boehm 81].

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 thousand delivered source instructions (KDSI).
- The intermediate COCOMO'81 model computes software development effort as a function of program size and a set of fifteen "cost drivers" that include subjective assessments of product, hardware, personnel, and project attributes.
- The advanced or 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

1. *development effort* : $MM = a * KDSI^{b}$

   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%.
2. *effort and development time* (TDEV) : $TDEV = 2.5 * MM^{c}$

The coefficents a, b and c depend 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

Table 1: development modes

1.1 BASIC COCOMO

The basic COCOMO applies the parameterised equation without much detailed consideration of project characteristics.

$MM = a * KDSI^{b}$
$TDEV=2.5 * MM^{c}$

Basic COCOMO	**a**	**b**	**c**
Organic	2.4	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

Overview 1: COCOMO I; equations and parameters of the basic COCOMO I

1.2 INTERMEDIATE COCOMO

The same basic equation for the model is used, but **fifteen** cost drivers are rated on a scale of 'very low' to 'very high' to calculate the specific effort multiplier and each of them returns an adjustment factor which multiplied yields in the total EAF (Effort Adjustment Factor). The adjustment factor is 1 for a cost driver that's judged as normal.

In addition to the EAF, the model parameter "a" is slightly different in Intermediate COCOMO from the basic model. The parameter "b" remains the same in both models.

$$MM = a * KDSI^{b}$$

$$TDEV = 2.5 * MM^{c}$$

Intermediate COCOMO	a	b	c
Organic	**3.2**	1.05	0.38
Semi-detached	**3.0**	1.12	0.35
Embedded	**2.8**	1.20	0.32

Overview 2: COCOMO I; equations and parameters of the intermediate COCOMO I

Man Month correction is now:

$$MM_{Korr} = EAF * MM_{nominal}$$

Equation 1: intermediate COCOMO; man month correction

Note: Only the Intermediate form has been implemented by USC in a calibrated software tool.

1.3 ADVANCED, DETAILED COCOMO

The Advanced COCOMO model computes effort as a function of program size and a set of cost drivers weighted according to each phase of the software lifecycle. The Advanced model applies the Intermediate model at the component level, and then a phase-based approach is used to consolidate the estimate [Fenton, 1997].

The four phases used in the detailed COCOMO model are: requirements planning and product design (RPD), detailed design (DD), code and unit test (CUT), and integration and test (IT). Each cost driver is broken down by phases as in the example shown in Table 2.

Cost Driver	Rating	RPD	DD	CUT	IT
ACAP	Very Low	1.80	1.35	1.35	1.50
	Low	0.85	0.85	0.85	1.20
	Nominal	1.00	1.00	1.00	1.00
	High	0.75	0.90	0.90	0.85
	Very High	0.55	0.75	0.75	0.70

Table 2: Analyst capability effort multiplier for detailed COCOMO

Estimates for each module are combined into subsystems and eventually an overall project estimate. Using the detailed cost drivers, an estimate is determined for each phase of the lifecycle.

1.4 ADA COCOMO

COCOMO has been continued to evolve and improve since its introduction. Beneath the "ADA COCOMO" model:

The model named "ADA_87" assumes that the ADA programming language is being used. The use of ADA made it easier to develop highly reliable systems and to manage complex applications. The APM_88 model is based upon a different "process model" called the ADA Process Model. Among the assumptions are:

- ADA is used to produce compiler checked package specifications by the Product Design Review (PDR)
- Small design teams are used
- More effort is spent in requirements analysis and design
- Less effort is spent in coding, integration, and testing

The APM_88 model incorporates changes to the equations, several cost drivers, and a variety of other tables. For further information please contact [Boehm 81].

1.5 EXAMPLE OF INTERMEDIATE COCOMO

Required: database system for an office automation project. Project = organic (a=3.2, b = 1.05, c=0.38), 4 modules to implement:

data entry	0.6 KDSI
data update	0.6 KDSI
query	0.8 KDSI
report generator	1.0 KDSI
System SIZE	3.0 KDSI

Efforts are rated as follows (all others nominal, 1.0):

cost drivers	level	EAF
complexity	high	1.15
storage	high	1.06
experience	low	1.13
prog capabilities	low	1.17

$MM_{Korr} = (1.15*1.06*1.13*1.17) * 3.2 * 3.0^{1.05}$

$MM_{Korr} = 1.61 * 3.2 * 3.17$

$MM_{Korr} = 16.33$

$TDEV = 2.5 * 16.33^{0.38}$

TDEV = 7.23 (>7months to complete)

How many people should be hired?

MM_{Korr} / TDEV = team members

16.33 / 7.23 = 2.26 (>2 team members)

1.6 ADVANTAGES OF COCOMO'81

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

1.7 DRAWBACKS OF COCOMO'81

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

CHARTER 2: COCOMO II, COCOMO 2000

2 COCOMO II

At the beginning an overall context is given where the need of the reengineering of COCOMO I is stressed as well as the focused issues for the new COCOMO version are presented.

2.1 REENGINEERING COCOMO I NEEDS

- new software processes
- new phenomena: size, reuse
- need for decision making based on incomplete information

2.2 FOCUSED ISSUES

The reengineering process focussed on issues such as:

1. non-sequential and rapid-development process models
2. reuse-driven approaches involving commercial-off-the-shelf (COTS) packages
3. reengineering [reused, translated code integration]
4. applications composition
5. application generation capabilities
6. object oriented approaches supported by distributed middleware
7. software process maturity effects
8. process-driven quality estimation

2.3 STRATEGY

1. Preserve the openness of the original COCOMO
2. Key the structure of COCOMO II to the future software marketplace sectors
3. Key the inputs and outputs of the COCOMO II submodels to the level of information available
4. Enable the COCOMO II submodels to be tailored to a project's particular process strategy (early prototyping stage [application composition model], Early Design stage, post-architecture stage

COCOMO II provides a family (COCOMO suite) of increasingly detailed software cost estimation models, each tuned to the sectors' needs and type of information available to support software cost estimation [Boehm 2000b].

2.4 THREE PRIMARY PREMISES

PARTICULAR PROCESS DRIVERS: current and future software projects tailor their processes to their particular process drivers (no single, preferred software life cycle model anymore), process drivers include COTS /reusable software availability, degree of understanding requirements and architectures and other schedule constraints as size and required reliability

INFORMATION: Consistency between granularity of the software model and the information available

RANGE ESTIMATES: coarse-grained cost driver information in the early project stages, and increasingly fine-grained information in later stages. Not point estimates of software cost and effort, but rather range estimates tied to the degree of definition of the estimation inputs.

2.5 DIFFERENCES BETWEEN COCOMO I AND COCOMO II

The major differences between COCOMO I AND COCOMO II are:

- COCOMO'81 requires software size in KDSI as an input, but COCOMO II is based on **KSLOC** (logical code). The major difference between DSI and SLOC is that a single Source Line of Code may be several physical lines. For example, an "if-then-else" statement would be counted as one SLOC, but might be counted as several DSI.
- COCOMO II addresses the following **three phases** of the spiral life cycle: applications development, early design and post architecture
- 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.
- The estimation equation exponent is determined by **five scale factors** (instead of the three development modes)
- Changes in cost drivers are:
  - Added cost drivers (7): DOCU, RUSE, PVOL, PLEX, LTEX, PCON, SITE
  - Deleted cost drivers (5): VIRT, TURN, VEXP, LEXP, MODP
  - Alter the retained ratings to reflect more up-do-date software practices
- Data points in COCOMO I: 63 and COCOMO II: 161
- 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

3 COCOMO II MODEL DEFINITION

COCOMO II provides three stage series of models for estimation of software projects:

APPLICATION COMPOSITION MODEL for earliest phases or spiral cycles [prototyping, and any other prototyping occurring later in the life cycle].

EARLY DESIGN MODEL for next phases or spiral cycles. Involves exploration of architectural alternatives or incremental development strategies. Level of detail consistent with level of information available and the general level of estimation accuracy needed at this stage.

POST-ARCHITECTURE MODEL: once the project is ready to develop and sustain a fielded system it should have a life-cycle architecture, which provides more accurate information on cost driver inputs, and enables more accurate cost estimates.

In the following sections the Early Design and Post-Architecture models are presented. The Application Composition Model will be discussed in section 6.3 p. 16.

3.6 NOMINAL SCHEDULE ESTIMATION EQUATIONS (NS)

The Early Design and Post-Architecture model use the same approach for product sizing (including reuse) and for scale factors as well as for estimating the amount of effort (PM_{NS}) and calendar time it will take to develop ($TDEV_{NS}$) a software project. The formula of the nominal-schedule (NS) is given here:

$$PM_{NS} = A x Size^{E} x \prod_{i=1}^{n} EM_i$$

where A = 2.94 (for COCOMO II.2000) Equation 2: Person month

where $E = B + 0.01 x \sum_{j=1}^{5} SF_j$ where B = 0.91 (for COCOMO II.2000)

$$TDEV_{NS} = Cx(PM_{NS})^{F}$$

where $F = D + 0.2 x 0.01 x \sum_{j=1}^{5} SF_j$ Equation 3: Time to develop

where $F = D + 0.2 x (E-B)$ and C = 3.67 and D = 0.28

Inputs: A = Effort coefficient, E = scale factors (5) which account for relative economies or diseconomies of scale encountered for software projects of different sizes, EM = effort multipliers (n=7 for the Early Design model, n=17 for the Post-Architecture model). The values A, B, C and D can be calibrated to one's own database of projects. It is recommended that at least A and C is calibrated to the local development environment to increase the model's accuracy. The scale factors and cost drivers are discussed in 4 p. 11.

$$\text{Size} = \left(1 + \frac{\text{REVL}}{100}\right) \times (\text{New KSLOC} + \text{Equivalent KSLOC})$$

$$\text{Equivalent KSLOC} = \text{Adapted KSLOC} \times \left(1 - \frac{\text{AT}}{100}\right) \times \text{AAM}$$

$$\text{where AAM} = \begin{cases} \dfrac{\text{AA} + \text{AAF} \times (1 + [0.02 \times \text{SU} \times \text{UNFM}])}{100}, & \text{for AAF} \le 50 \\ \dfrac{\text{AA} + \text{AAF} + (\text{SU} \times \text{UNFM})}{100}, & \text{for AAF} > 50 \end{cases}$$

$$\text{AAF} = (0.4 \times \text{DM}) + (0.3 \times \text{CM}) + (0.3 \times \text{IM})$$

Equation 4: Sizing equations

Inputs: REVL = Requirements Evolution and Volatility (3.7.3 p. 10), New KSLOC = new source code in thousands of source lines of code, the rest of Equation 4 will be discussed in section 3.7.1 A Reuse Model p. 9.

Economies / Diseconomies of scale: the scale factors (SF) reflect this. If E < 1.0 the project exhibits economies of scale, if E = 1.0 it is in balance, if E > 1.0 the project exhibits diseconomies of scale, due to two main factors: growth of interpersonal communications overhead and growth of large system integration overhead.

3.7 CODE CATEGORY OR DISAMBIGUATION

For the determination of size code is the basis. But code can have different sources. These sources are briefly discussed and some guidelines for quantifying software are given Table 3 summarizes this excursion.

New code: new code written manually (without any ICASE; for ICASE see 6.3 Applications Composition p. 16) different worded: from scratch

Reused code: pre-existing code that is treated as a *black-box* and plugged into the product as it is

Adapted code: pre-existing code that is treated as a *white-box* and is modified for use with the product

COTS component: commercial off-the-shelf component, leased, licensed, source code is not available

Automatically translated code: pre-existing code for which translation is supported by automated tools (keyword: reengineering/conversion)

The effective size of reused and adapted code is adjusted to be its equivalent in new code (ESLOC = equivalent source lines of code). The Adjustment is based on the additional effort it takes to modify the code for inclusion in the product. Additional costs for software understanding arise as soon as the unmodified (black-box) code is modified (white-box). For the adapted code the programmer familiarity with the code is also taken into account. The following section uses the expression reuse for a generic term (reuse and adapted code).

3.7.1 A Reuse Model

An analysis of the NASA Software Engineering Laboratory of reuse costs indicates that the amount of modification of the reused code to the resulting cost to reuse is nonlinear in three ways:

I) The effort required to reuse code does not start at zero because of the 'fix costs' for assessing, selecting and assimilating the reusable component (ca. 5%)

II) The cost of understanding the software to be modified and the programmer familiarity

III) The relative cost of checking module interfaces

The size of II) and III) can be reduced by good software structuring (modular, hierarchical, explained).

Note: Small modifications generate disproportionately large costs.

The following equation is used to count the module interfaces N to be checked if a module k is modified out of m software modules. Equation 5

$$N = k \times (m-k) + k \times ((k-1)/2)$$

As previously mentioned the COCOMO II treats software reuse as nonlinear. This involves estimating the amount of software to be adapted and three degree-of-modification factors: percentage– of design modified (DM), - of code modified (CM), - of integration effort required for adapted software (IM). The detailed ratings can be found in [Boehm et al. 2000a].

Further increments are:

- percentage of software understanding increment (SU), which consists of three sub-factors: structure, application clarity, self-descriptivness. If these sub-factors are rated as 'very high' SU is 10%, as they are rated 'very low', SU is 50%.
- degree of Assessment and Assimilation (AA), which states the appropriation to the application and the integration (documentation, test and evaluation modules available?) into the overall product
- degree of programmers unfamiliarity (UNFM), which is 0.0 if the programmer works with the software every day.

The equation (Equation 4) can be found in section 3.68. Note: at the worst case, it can take twice the effort to modify a reused module than it takes to develop it as new! The best case follows a one for one correspondence between adapting an existing product and developing it from scratch.

3.7.2 Automatically translated code

This is an additional refinement for the COCOMO II reuse model to estimate the costs of software reengineering and conversion where automated tools are used for software restructuring. It is seen as a separate activity from development.

$$PM\ Auto = (Adapted\ SLOC\ (AT/100)) / ATPROD$$

Equation 6: automatically translated code

PM Auto = Estimated effort of the automated translation

ATPROD = default productivity value for automated translation is 2400 source statements per person month. This value can vary with different technologies.

AT = percentage of the code that is re-engineered by automatic translation. It is a strong function of the difference between the boundary conditions (e.g., use of COTS packages, change from batch to interactive operation) of the old code and the re-engineered code.

3.7.3 Requirements Evolution and Volatility (REVL)

COCOMO II uses a factor called REVL, to adjust the effective size of the product caused by requirements evolution and volatility, by such factors as mission or user interface evolution, technology upgrades or COTS volatility. It is the percentage of code discarded due to requirements evolution. Size $_D$ in the following equation is the reuse-equivalent of the delivered software.

$$Size = (1 + (REVL/100)) \times Size_D$$

Equation 7: requirements evolution and volatility

CODE CATEGORY		**reused**	**adapted**	**automatically translated**	**COTS component**	**new (ICASE)**	**new**
approach		black-box	white box & modifications		black-box	scratch	scratch
effort estimation via		SLOC or UFP	SLOC or UFP	separate translation productivity rate	special	Application Points	SLOC or UFP
adjustments to new code		yes	yes	no separate equation	no separate equation	no separate equation	yes
included in nominal-schedule (NS)		yes	yes	no	no (see 6.2 COCOTS p. 15)	no (see 6.3 Applications Composition p. 16)	yes
note					special form of reused code		
annotation					new glue code is associated with COTS component		
Reuse Parameters	DM	0%	0%-100% normally>0%	0%	0%	not applicable	not applicable
	CM	0%	0%-100% usually > DM and must be > 0%	0%	0%		
	IM	0%-100% rarely 0%, but could be very small	0%-100% IM usually moderate & can be > 100%	0%-100%	0%-100%		
	AA	0%-8%	0%-8%	0%-8%	0%-8%		
	SU	not applicable	0%-50%	not applicable	not applicable		
	UNFM		0-1				

Table 3:code types or disambiguation

4 DEVELOPMENT EFFORT ESTIMATES

4.1 SIZING

A good size estimate is very important for a good model estimate. However determining size can be very challenging because projects are generally composed of new, reused (with or without modifications) and automatically translated code. The baseline size in COCOMO II is a count of new lines of code.

SLOC is defined such that only source lines that are DELIVERED as part of the product are included -- test drivers and other support software (-as long as they aren't documented as carefully as source code- is excluded).

One SLOC is one logical source statement of code (e.g. declarations are counted, comments not)

In COCOMO II effort is expressed as person months (PM). Person month is the amount of time one person spends working on the software development project for one month. Code size is expressed in thousands of source lines of code (KSLOC). The goal is to measure the amount of intellectual work put into program development. Counting source lines of Code (SLOC) takes account of new lines of code (reused code has to be adjusted). There are two possibilities: either to count the source lines of code (with the Software Engineering Institute (SEII) checklists or tool supported (see 10 Tools p. 19)) or to count the unadjusted function points and to convert them via "backfiring" tables to source lines of code. For further information on counting the unadjusted function points (UFP) as well as the ratios for the conversion can be found in [Boehm et al. 2000a] and [conversion tables for SLOC/UFP].

4.2 THE SCALE FACTORS

Most significant input to the COCOMO II model is size. Size is treated as a special cost driver in that it has an exponential factor, E. This exponent is an aggregation of five *scale factors*. They are only used at the project level for both the Early Design and the Post-Architecture model. All scale factors have qualitative rating levels ('extra low' to 'extra high'). Each scale factor is the subjective weighted average of its characteristics. The ratings can be fount at [Boehm et. al 2000a].

The five Scale Factors are:

1. PREC Precedentedness (how novel the project is for the organization)
2. FLEX Development Flexibility
3. RESL Architecture / Risk Resolution
4. TEAM Team Cohesion
5. PMAT Process Maturity

Scale Factors (W_i)		annotation
PREC	If a product is similar to several previously developed project, then the precedentedness is high	Describe much the same influences that the original Development Mode did, largely intrinsic to a project and uncontrollable
FLEX	Conformance needs with requirements / external interface specifications, ...	
RESL	Combines Design Thoroughness and Risk Elimination (two scale factors in Ada).	Identify management controllables by which projects can reduce diseconomies of scale by reducing sources of project turbulence, entropy and rework.
TEAM	accounts for the sources of project turbulence and entropy because of difficulties in synchronizing the project's stakeholders.	
PMAT	time for rating: project start. Two ways for rating: 1. by the results of an organized evaluation based on the SEI CMM, 2. 18 Key Process Areas in the SEI CMM.	

Table 4: scale factors description for COCOMO II

Scale Factors for COCOMO II Early Design and Post-Architecture Models

Scale Factors (W_i)	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	throughly familiar
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
RESL	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
PMAT	Weighted average of "Yes" answers to CMM Maturity Questionnaire					

Table 5: Scale factors for COCOMO II

4.3 COST DRIVERS

COCOMO II has 7 to 17 multiplicative factors that determine the effort required to complete a software project. All cost drivers have qualitative rating levels ('extra low' to 'extra high') that express the impact of the driver and a corresponding set of effort multiplier. The nominal level always has an effort multiplier (EM) of 1.00, which does not change the estimated effort. So a cost driver's qualitative rating is translated into a quantitative one for use in the model. The COCOMO II model can be used to estimate effort and schedule for the whole project or for a project that consists of multiple modules. The size and cost driver ratings can be different for each module, with the exception of the Required Development Schedule (SCED) cost driver and the scale factors.

In the Early Design model a reduced set of multiplicative cost drivers is used as shown in Table 6. The early cost drivers are obtained by combining the Post-Architecture model cost drivers.

For example, if a project will develop software that controls an airplane's flight, the Required Software Reliability (RELY) cost driver would be set to 'very high'. That rating corresponds to an effort multiplier of 1.26, meaning that the project will require 26% more effort than a typical software project.

	Early Design cost drivers	**Post-Architecture cost drivers (Counterpart combined)**
Product reliability and complexity	RCPX	RELY, DATA, CPLX, DOCU
Required reuse	RUSE	RUSE
Platform difficulty	PDIF	TIME, STOR, PVOL
Personnel capability	PERS	ACAP, PCAP, PCON
Personnel experience	PREX	AEXP, PEXP, LTEX
Facilities	FCIL	TOOL, SITE
Required Development Schedule	SCED	SCED

Table 6: Effort Multipliers for the Early Design and Post-Architecture

4.4 COMPARISON OF THE EARLY DESIGN MODEL AND POST-ARCHITECTURE MODEL

	Early Design model	Post-Architecture model
deployment	This model is used to make rough estimates of a project's cost and duration before its entire architecture is determined.	This is the most detailed COCOMO II model. It is used after project's overall architecture is developed. This stage proceeds most cost-effectively if a software life-cycle architecture has been developed, validated with respect to the systems mission, concept of operation and risk.
information available	not enough	fine-grain cost estimation can be supported
cost drivers	set of seven cost drivers	set of 17 multiplicative cost drivers grouped into four categories (Product factors, Platform factors, Personnel factors, Project factors). These four categories are parallel the four categories of COCOMO'81. Many of the seventeen factors of COCOMO'II are similar to the fifteen factors of COCOMO'81. For added or removed cost driver see section 2.5 p. 7.
involves ...	exploration of alternative software/system architecture and concepts of operation	actual development and maintenance of a software product

Table 7: comparison of the Early Design model and Post-Architecture model

5 BAYESIAN APPROACH

In short terms the Bayesian approach consists of merging expert-opinion (Delphi) and project data, based on the variance of the two sources of information, in order to determine a more robust posterior model.

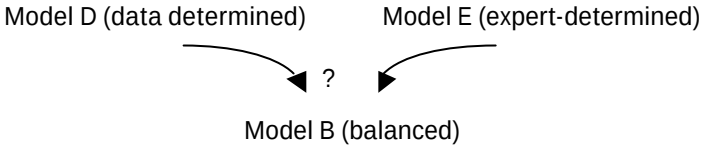
Model D (data determined) has all its cost-driver values calibrated to the best data that is available.

Model E (expert-determined) uses the same form and cost drivers as Model D, but all cost driver values are determined by expert consensus.

Two problems arise:

- Problem of Model D (best data available): project data is inevitably collected with imprecise definitions of what is included in the product, process and workforce.
- Problem of Model E: some of the cost driver values calibrated to the noisy data do not square with the expert's experience.

For example: For a new project Model D estimates 122 mm, Model E comes to 236 mm. which model is to choose? if the average should be taken then just the halfway estimate 179 mm is correct?



Model B (balanced) is constructed in a way that it favours experts for cost drivers where they are in strong agreement and the data fit is week, and favours the data fit for cost drivers where it is strong and the experts disagree. This technique is provided by the Bayesian approach to determine model parameters.

Model E (Expert-Determined) parameter values and their variances are taken as the *a priori* knowledge about the parameter values.

Model D (data-Determined) parameter values and their variances are taken as *new information* which can be used to determine a posteriori update of the parameter values.

The Bayesian approach basically produces a weighted average of the Model D and E values, which gives higher weights to parameter values with smaller variances.

Wideband Delphi process guaranties the consistency of interpretations and increased consensus among the experts. Ends inevitably with some variance around the experts' mean parameter values, indicating the degree to which the experts disagree on the parameter's effect on the quantity being estimated.

5.5 COCOMO II MODELING METHODOLOGY

Preparation includes: Parameter definition, how to use parameters to produce effective estimates. The methodology tries to minimize risk of lost expert time and maximize estimates. Here the question arises which parameters are significant (most estimate efficient), in which way and with which rating scale.

5.5.1 Seven modeling steps

1. analyse existing literature
2. review software cost modelling literature
3. to insight on improved functional forms potentially significant parameters
4. parameter definition issues (e.g. size)
5. identification of potential new parameters Process Maturity and Multisite Development
6. continuation of a number of parameters from COCOMO I
7. dropping of such COCOMO I parameters as turnaround time and modern programming practices (subsumed by process maturity)

The Bayesian approach was used for COCOMO II and is (will be) reused for COCOTS, COQUALMO, COPSEMO and CORADMO.

5.6 THE ROSETTA STONE

The *ROSETTA STONE* is a system that updates COCOMO I so that it can be used with COCOMO II models. The *ROSETTA STONE* permits users to translate project files from COCOMO 81 to COCOMO II (backwards compatibility).

6 EMERGING EXTENSIONS

Because of the rapid changes in software engineering development not all directions of impact could take place in the COCOMO II model. So some emerging extensions were required to overcome the deficiencies. All of them are complementary to the COCOMO II model and some of them are still experimental, their calibration and counting rules aren't robust enough till now. Further research still has to be done.

In the following section they are presented briefly. For further particulars please contact the list of references. The discussed extensions are:

- estimating the cost of software COTS integration (COCOTS)
- Application Composition Model
- phase distributions of schedule and effort (COPSEMO)
- rapid application development effort and schedule adjustments (CORADMO)
- quality in terms of delivered defect density (COQUALMO)
- effects of applying software productivity strategies / improvement (COPROMO)
- System Engineering (COSYSMO)

6.1 STATUS QUO

The table beneath tries to give an overview of the current status of some of the discussed extension of COCOMO II. For the ranking of the column 'status quo' the following steps are taken as basis: Delphi round 1 →Delphi round 2→gather Data→analyse→Bayesian Analysis→Calibrate Model [Fakharzadeh 2001-b].

Extension	Status quo	Still experi-mental	Tool supported	New cost drivers	Annotation
Application Composition Model		yes		no	calibration and counting rules not robust till now

Extension	Status quo	Still experi-mental	Tool supported	New cost drivers	Annotation
COCOTS	not fully formulated and validated	yes	not up to date	no	does not treat the long-term operation and maintenance
COPSEMO	about to complete Delphi round 1 [Fakharzadeh 2001-a]	yes	yes	no	basis for other COCOMO II model extensions
CORADMO	Delphi round 2 completed [Fakharzadeh 2001-b]	yes	yes	yes	

Table 8: status quo of the emerging extensions

6.2 COCOTS

[COnstructive COTS] focuses on estimating the cost, effort, and schedule associated using commercial off-the-shelf (COTS) components in a software development project. It captures costs that traditionally have been outside the scope of COCOMO. A COTS product refers to pre-built, commercially available software components that are becoming important in the creation of new software systems. The sense in using COTS components is that it requires less development time by taking advantage of existing, market-proven, vendor-supported products and reduces overall system development costs.

A definition of COTS components might be:

> commercial software product - sold, leased, licensed at advertised prices, source code unavailable, usually periodic releases with feature growth (obsolescence), future development not under control of application developer

As a consequence of this definition, the following COTS phenomena obtain: no control over a COTS product's functionality or performance, most COTS products are not designed to interoperate with each other, no control over a COTS product's evolution, COTS vendor behavior vary widely.

This in turn leads to some basic risks inherent using COTS components: immaturity of the product and/or the vendor, inexperience of integrators and/or users with the product, incompatibility of the product with the larger application, platform, or other COTS components in the system, lack of control over the product's current and future functionality (COTS is evolutionary (volatility)), customer/user view that COTS integration is *Plug-and-Play* (i.e. the assumption that use of COTS means zero development time).

These risks can be reduced or eliminated using the following mitigation strategies: qualification testing, benchmarking, reference checking, compatibility analysis.

If risks are managed using COTS, COTS can be the right solution (most cost effectively and shortest schedule approach). COTS is a feasible solution when it is in balance with the three determinants: feasibility technical economic and strategic constraints.

The four submodels

To estimate the total effort of a COTS component integration an approach with four submodels (Assessment, Tailoring, Glue Code, Volatility) is followed.

ASSESSMENT selection regarding / balancing between quality/requirements/aspects
- functional requirement (capability offered)
- performance requirement (timing and sizing constraints)
- nonfunctional requirements (cost/training/installation/maintenance/reliability)

TAILORING configuration for use in a specific context in respect to complexity

GLUE CODE glueware/binding code/new code which is required to get a COTS product integrated into a larger system

VOLATILITY is the additional effort that results from the impact on the larger system of the effects of swapping COTS components out of the system with newer version of those components that have been released by the COTS vendors

The total effort is defined as follows:

$$\begin{aligned} \textit{Total Effort} &= \textit{Assessment Effort} + \textit{Tailoring Effort} + \\ &\quad \textit{Glue Code Effort} + \textit{System Volatility} \\ \textit{Assessment Effort} &= \textit{Filtering Effort} + \textit{Detailed Assessment Effort} \end{aligned}$$

Equation 8: COCOTS effort estimation

The model is still immature.

6.3 APPLICATIONS COMPOSITION

Application composition methods are more and more used. These methods rely on having an Integrated Computer-Aided Software Engineering (ICASE) environment to accelerate product development. ICASE includes generally following capabilities:

- an application framework (e.g. client-server, peer-to-peer,..) with middleware to integrate and manage the execution of the applications components
- a set of common utilities (GUI builder, DBMS, Network support packages)
- domain architecture and set of reusable domain components, repository
- tools for design, construction, integration and test.

Traditionally applications requiring 10-20 people for one to two years using application composition methods. These can be developed by 2-6 people in 2-6 months. Note: project efforts tend to be relatively small <72 PM.

Because there is no way to size products in SLOC or FP (too low-level) research was done to find a [passend, suitable] size-metric. Object Points which was developed by Banker, Kauffman, Kumar in 1991 was found. In the Object Point approach numbers of screens, reports and third-generation language (3GL) modules for basic sizing primitives are used. The function point convention was followed of weighting these by complexity and adding them together to obtain an overall size metric, which then was adjusted for reuse.

The Object Point approach was adapted with two changes:

1. rating scales for determining a project's productivity rate in NAP[1]/PM in terms of ICASE system maturity and capability and the developer's experience and capability in using the ICASE
2. rename Object Points in Application Points

Objects/Applications can be determined by the standard definitions of screens, reports and 3GL components in the ICASE environment. Then each object is classified into simple, medium and difficult complexity levels depending on values of characteristic dimensions defined. For further information see [Boehm et al. 2000a].

6.4 COPSEMO

[Constructive Phase Schedule & Effort Model] It has been observed that the COCOMO II's duration calculation which presently implements a waterfall process model seems unreasonable for small projects, those with effort under two person years. Therefore COPSEMO focuses on the cost of developing software (below 64 person months) by using a more complex calculation for the low effort situations and by applying the lifecycle anchoring concepts discussed by [Boehm 96]. COPSEMO model assumes that data is available form a COCOMO II model. It has no drivers per se. The model allows to be applied to the different phases by specifying the effort and schedule in the appropriate percentages.

COPSEMO uses the MBSE and Rational Unified Process Milestones. Three milestones separate the effort and schedule phases of COPSEMO: Life Cycle Objectives review (LCO), Life Cycle Architecture review (LCA) and Initial Operational Capability (IOC). Project Start (PS) and Project Release (PR) serve as start and endpoints. Further information on the MBSE and COPSEMO itself can be found in [Boehm et al. 2000a] and [Brown 2000].

[1] NAP = percentage of reuse which is expected to be achieved in this project (100% reuse)/100

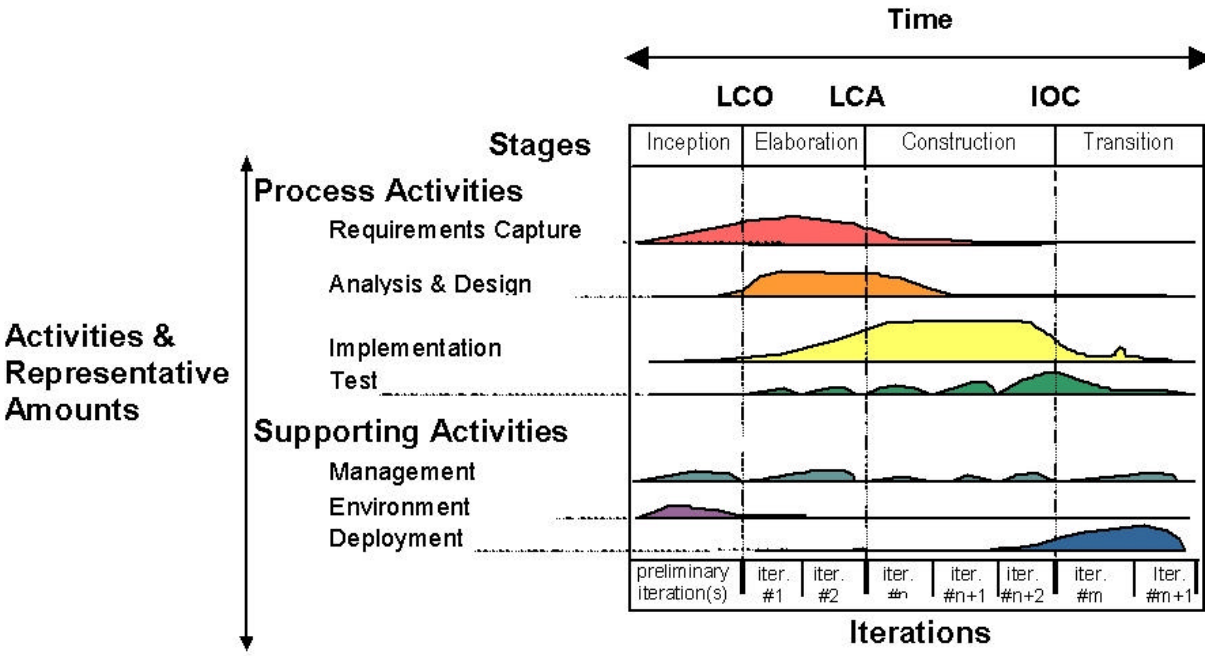


Figure 1: Activity Levels

COCOMO II was calibrated for minimum 2000 SLOC. Giving an example: By having all drivers nominal a project of 4700 SLOC is estimated with COCOMO II TDEV to take 16.1 PM over 8.9 calendar months with a staff of 1.8 persons. In reasonable engineering project management 4 people would handle the project in 4 calendar months. The lower bound with 16 person months and 4 months was selected. The upper bound , 64 PM was selected by incrementally increasing the calendar months until COCOMO II produced a reasonable TDEV and staff estimate.

While actually developed as part of the early CORADMO models, COPSEMO is now recognized as a separate, independent model that will be used as the basis for other COCOMO II model extensions.

6.5 CORADMO

[Constructive Rapid Application Development Model] Focuses on the cost of developing software using rapid application development (RAD) techniques, because not any RAD strategy is addressed in COCOMO II.

RAD refers to an application of any number of techniques or strategies to reduce software development cycle time and sometimes effort as well.

The intent of the CORADMO is to calculate/predict the schedule (months, M), personnel (P), and adjusted effort (person months, PM) based on the distribution of effort and schedule to the various stages, and impacts of the selected schedule driver ratings on the M, P, and PM of each stage.

There are different types of RAD differentiated by B. Boehm [Boehm 99]: Generator RAD (GRAD), Composition RAD (CRAD), Full-System RAD (FRAD) and Dumb RAD (DRAD). The differences between these RADs are not discussed here. CORAMO applies to the GRAD, CRAD and FRAD but NOT to DRAD.

New Drivers A set of five drivers reflect identifiable behavioural characteristics. These drivers are: Reuse and very high-level Languages (RVHL), Development Process Reengineering (DPRS), Collaboration Efficiency (CLAB), Architecture, Risk Resolution (RESL), Pre-positioning Assets (PPOS).

The Relation to the COCOMO II is that CORADMO applies as a base for COPSEMO (Pre-processor model). CORADMO excludes COCOTS and COQUALM.

Future work: new cost drivers whose additional effect on schedule may be significant will be included (such as personnel experience, capability and continuity factors as well as requirements volatility).

6.6 COQUALMO

[COnstructive QUALity Model] In software estimation, it is important to recognize the strong relationships between cost, schedule and quality. They form three sides of the same triangle. Beyond a certain point (the "Quality is Free" point), it is difficult to increase software quality without increasing either the cost or schedule or both for the software under development. Similarly, development schedule cannot be drastically compressed without hampering the quality of the software product and/or increasing the cost of development. Software estimation models can play an important role in facilitating the balance of these three factors. COQUALMO is such an estimation model.

6.7 COPROMO

[Constructive Productivity Improvement Model] Focuses on predicting the most cost effective allocation of investment resources in new technologies intended to improve productivity.

6.8 EXPERT COCOMO

Expert COCOMO was developed by Dr. Ray Madachy and uses heuristics to flag potential risks found in specified project development conditions. It aids planning projects by identifying, categorizing, quantifying and prioritising projects risks. It also detects cost estimate input anomalies. Further information can be found in [Expert COCOMO] or [Boehm et al. 2000a].

6.9 COSYSMO

[Constructive Systems Engineering Cost Model] The purpose of the COSYSMO is to estimate the System Engineering (SE) tasks in software-intensive projects. The CSE Research Group Selected ANSI/EI632 SE standard as a guide for identifying the tasks addressed in COSYSMO. The focus of the initial increment of the model is on the costs of SE in Information Processing (IP) subsystems, hence the naming of COSYSMO-IP. Several CSE Affiliates and members of the International Council on Systems Engineering (INCOSE) have been involved in the definition of the drivers and strategic direction of the model.

7 OUTLOOK

The target is to improve the calibration of COCOMO II and of the tool itself. The current plan is to release a new calibration annually. It is anticipated that USC COCOMO II.2001.0 will indeed have a new calibration. Finally, experience has shown that if an organization calibrates the multiplicative constant in COCOMO II to its own empirical data, the accuracy of the model can be greatly improved over the generic calibration results.

8 PROBLEMS WITH EXISTING MODELS

There is some question as to the validity of existing algorithmic models applied to a wide range of projects. It is suggested that a model is acceptable if 75 percent of the predicted values fall within 25 percent of their actual values (Fenton, 1997). The reasons why existing modeling methods have fallen short of their goals include model structure, complexity, and size estimation.

Structure: Models based on limited data sets tend to incorporate the particular characteristics of the data. This results in a high degree of accuracy for similar projects, but restricts the application of the model.

Complexity: An organization's particular characteristics can influence its productivity [Humphrey, 1990]. The cost drivers are extremely subjective. It is difficult to ensure that the factors are assessed consistently and in the way the model developer intended.

9 CRITICS

ADVANTAGES:

- COCOMO II is an industry standard
- very profound information is easy available
- clear and effective calibration process by combining delphi with algorithmic cost estimation techniques (Bayesian method)
- 'backwards compatibility' with the Rosetta Stone
- various extension for almost every purpose are available
- Tool support (also for the various extensions)

DRAWBACKS:

- The 'heart' of COCOMO II is still based on a waterfall process model (predilection)
- most of the extensions are still experimental and not fully calibrated till now
- duration calculation for small projects is unreasonable

10 TOOLS

The software implementation of the model follows a specific naming convention. The theoretical model is referred to as COCOMO II. The USC-CSE software implementation of the model is referred to as USC COCOMO II, to distinguish it from other academic or commercial implementations of the model. Here are several available Tools which support COCOMO II itself or its extensions. The list is not complete.

10.10 COMMERCIAL COCOMO II SOFTWARE

COSTAR	Softstar Systems	Amherst, NH	http://www.softstarsystems.com/
Cost Xpert	Cost Xpert Group	San Diego, CA	http://www.costxpert.com/
ESTIMATE Professional	Software Productivity Centre	Vancouver, BC, Canada	http://www.spc.ca/

Table 9: Commercial COCOMO II Software

10.11 FREE COCOMO II SOFTWARE (EXTENSIONS)

COPSEMO	Stand-alone spreadsheet implementation, Microsoft Excel, http://sunset.usc.edu/research/COPSEMO/copsemo.xls
CORADMO	http://sunset.usc.edu/research/CORADMO/CORADMO.ZIP (Microsoft Excel etc.)
COCOTS	http://sunset.usc.edu/research/COCOTS/tools/COCOTSglueV20.zip (Microsoft Excel etc.)
expert COCOMO	http//sunset.usc.edu/COCOMO II/expert_COCOMO/expert_COCOMO.html.
CODECOUNT	http://sunset.usc.edu/research/CODECOUNT/download.html

Table 10: Free USC-CSE COCOMO II Software

V. CONCLUSION

Software cost estimation is an important part of the software development process. The COCOMO suite (COCOMO II model and its extensions) offers a powerful instrument to predict software costs. Unfortunately not all of the extensions are already calibrated and therefore still experimental. Only the Post-Architecure model is implemented in a calibrated software tool. Despite this disadvantage the COCOMO II suite helps managing software projects. It supports process improvement analyses, tool purchases, architecture changes, component make/buy tradeoffs and decision making process with credible results. Many endeavours were done to measure up to the changes in software life cycles, technologies, components, tools, notations and organizational cultures since the first version of COCOMO (COCOMOI, COCOMO 81).

VI. LIST OF REFERENCES

Boehm 81	Boehm, B.W. (1981). *Software Engineering Economics*. Prentice Hall.
Boehm 96	Boehm, B.W. (1996). *Anchoring the Software Process,* IEEE Software, 13, 4 July 1996, pp. 73-82.
Boehm 99	Boehm, B.W. (1999). *Making RAD Work for your Project*, Extended version of March 1999, IEEE Computer Column; USC Technical Report USC-SCE-99-512.
Boehm et al. 2000a	Boehm, B.W. et al. (2000). Software Cost Estimation with COCOMO II, Prentice Hall PTR; 1st edition (January 15, 2000).
Boehm et al. 2000b	Boehm, B.W., Abts, C., Clark, B., and Devnani-Chulani. S. (2000). *COCOMO II Model Definition Manual*. Version 2.1, The University of Southern California.
Brown 2000	A. Winsor Brown 2000. COPSEMO Summary. USC Center for Software Engineering, V2.0 05/18/2000.
Fakharzadeh 2001-a	Fakharzadeh, Cyrus (2001). *CORADMO Update*, Presentation, Cyrus Fakharzadeh, CSE Technology Week, February 2001.
Fakharzadeh 2001-b	Fakharzadeh, Cyrus (2001). *CORADMO in 2001: A RAD Odyssey,* Presentation, Cyrus Fakharzadeh, 16th International Forum on COCOMO and Software Cost Modeling, October 2001.
Fenton 1997	Fenton, N.E. and Pfleeger, S.L. (1997). *Software Metrics: A Rigorous and Practical Approach.* International Thomson Computer Press.
Humphrey 1990	Humphrey, W.S. (1990). *Managing the Software Process*. Addison-Wesley Publishing Company.
IFPUG 1994	IFPUG (1994). *Function Point Counting Practices*: Manual Release 4.0, International Function Point User's Group, Blendonview Office Park, 5008-28 Pine Creek Drive, Westerville, OH 43081-4899.
COCOMO suite	Center for Software Engineering, University of Southern California http://sunset.usc.edu/research/cocomosuite/index.html
Conversion tables for SLOC/UFP	http://www.spr.com/library/0Langtbl.htm.
DACS	Data and Analysis Center for Software DACS http://www.dacs.dtic.mil/databases/url/key.hts?keycode=1
Expert COCOMO	http//sunset.usc.edu/COCOMO II/expert_COCOMO/expert_COCOMO.html.