COCOMO II

Model Definition Manual

Version 2.1

Table of Contents

Acknowledgements ii

Copyright Notice iii

Warranty iii

1. Introduction 1
   - 1.1 Overview 1
   - 1.2 Nominal-Schedule Estimation Equations 1

2. Sizing 3
   - 2.1 Counting Source Lines of Code (SLOC) 3
   - 2.2 Counting Unadjusted Function Points (UFP) 4
   - 2.3 Relating UFPs to SLOC 6
   - 2.4 Aggregating New, Adapted, and Reused Code 7
   - 2.5 Requirements Evolution and Volatility (REVL) 12
   - 2.6 Automatically Translated Code 13
   - 2.7 Sizing Software Maintenance 13

3. Effort Estimation 15
   - 3.1 Scale Factors 16
   - 3.2 Effort Multipliers 25
   - 3.3 Multiple Module Effort Estimation 39

4. Schedule Estimation 41

5. Software Maintenance 42

6. COCOMO II: Assumptions and phase/activity distributions 44
   - 6.1 Introduction 44
   - 6.2 Waterfall and MBASE/RUP Phase Definitions 45
   - 6.3 Phase Distribution of Effort and Schedule 49
   - 6.4 Waterfall and MBASE/RUP Activity Definitions 53
   - 6.5 Distribution of Effort Across Activities 61
   - 6.6 Definitions and Assumptions 66

7. Model Calibration to the Local Environment 68

8. Summary 71
   - 8.1 Models 71
   - 8.2 Rating Scales 73
   - 8.3 COCOMO II Version Parameter Values 75
   - 8.4 Source Code Counting Rules 77

Acronyms and Abbreviations 81

References 85

Acknowledgements

The COCOMO II model is part of a suite of Constructive Cost Models. This suite is an effort to update and extend the well-known COCOMO (Constructive Cost Model) software cost estimation model originally published in Software Engineering Economics by Barry Boehm in 1981. The suite of models focuses on issues such as non-sequential and rapid-development process models; reuse driven approaches involving commercial-off-the-shelf (COTS) packages, reengineering, applications composition, and software process maturity effects and process-driven quality estimation. Research on the COCOMO suite of models is being led by the **Director of the Center of Software Engineering at USC, Barry Boehm** and other researchers (listed in alphabetic order):

Chris Abts	Ellis Horowitz
A. Winsor Brown	Ray Madachy
Sunita Chulani	Don Reifer
Brad Clark	Bert Steece

This work is being 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.

The successive versions of the tool based on the COCOMO II model have been developed as part of a Graduate Level Course Project by several student development teams lead by Ellis Horowitz. The latest version, USC COCOMO II.2000, was developed by the following graduate student:

Jongmoon Baik

Copyright Notice

Warranty

This manual is provided “as is” without warranty of any kind, either express or implied, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. Moreover, the Center for Software Engineering, USC, reserves the right to revise this manual and to make changes periodically without obligation to notify any person or organization of such revision or changes.

1. Introduction

1.1 Overview

This manual presents two models, the Post-Architecture and Early Design models. These two models are used in the development of Application Generator, System Integration, or Infrastructure developments [Boehm et al. 2000]. The Post-Architecture is a detailed model that is used once the project is ready to develop and sustain a fielded system. The system should have a life-cycle architecture package, which provides detailed information on cost driver inputs, and enables more accurate cost estimates. The Early Design model is a high-level model that is used to explore of architectural alternatives or incremental development strategies. This level of detail is consistent with the general level of information available and the general level of estimation accuracy needed.

The Post-Architecture and Early Design models use the same approach for product sizing (including reuse) and for scale factors. These will be presented first. Then, the Post-Architecture model will be explained followed by the Early Design model.

1.2 Nominal-Schedule Estimation Equations

Both the Post-Architecture and Early Design models use the same functional form to estimate the amount of effort and calendar time it will take to develop a software project. These nominal-schedule (NS) formulas exclude the cost driver for Required Development Schedule, SCED. The full formula is given in Section 3. The amount of effort in person-months, PM_{NS}, is estimated by the formula:

$$PM_{NS} = A \times Size^{E} \times \prod_{i=1}^{n} EM_i$$
$$\text{where } E = B + 0.01 \times \sum_{j=1}^{5} SF_j \qquad \textbf{Eq. 1}$$

The amount of calendar time, $TDEV_{NS}$, it will take to develop the product is estimated by the formula:

$$TDEV_{NS} = C \times \left(PM_{NS}\right)^{F}$$
$$\text{where } F = D + 0.2 \times 0.01 \times \sum_{j=1}^{5} SF_j \qquad \textbf{Eq. 2}$$
$$= D + 0.2 \times (E - B)$$

The value of n, the number of effort multipliers, EM_i, is 16 for the Post-Architecture model effort multipliers, EM_i, and 6 for the Early Design model. SF_j stands for the exponential scale factors. The values of A, B, EM_1, …, EM_{16}, SF_1, …, and SF_5 for the COCOMO II.2000 Post-Architecture model are obtained by calibration to the actual parameters and effort values for the 161 projects currently in the COCOMO II database. The values of C and D for the

COCOMO II.2000 schedule equation are obtained by calibration to the actual schedule values for the 161 project currently in the COCOMO II database.

The values of A, B, C, D, SF_1, …, and SF_5 for the Early Design model are the same as those for the Post-Architecture model. The values of EM_1, …, and EM_6 for the Early Design model are obtained by combining the values of their 16 Post-Architecture counterparts; the specific combinations are given in Section 3.2.2.

The subscript NS applied to PM and TDEV indicates that these are the nominal-schedule estimates of effort and calendar time. The effects of schedule compression or stretch-out are covered by an additional cost driver, Required Development Schedule. They are also included in the COCOMO II.2000 calibration to the 161 projects. Its specific effects are given in Section 4.

The specific milestones used as the end points in measuring development effort and calendar time are defined in Section 6, as are the other definitions and assumptions involved in defining development effort and calendar time. Size is expressed as thousands of source lines of code (SLOC) or as unadjusted function points (UFP), as discussed in Section 2. Development labor cost is obtained by multiplying effort in PM by the average labor cost per PM. The values of A, B, C, and D in the COCOMO II.2000 calibration are:

A = 2.94 B = 0.91
C = 3.67 D = 0.28

Details of the calibration are presented in Section 7, which also provides formulas for calibrating either A and C or A, B, C, and D to one's own database of projects. It is recommended that at least A and C be calibrated to the local development environment to increase the model's accuracy.

As an example, let's estimate how much effort and calendar time it would take to develop an average 100 KSLOC sized project. For an average project, the effort multipliers are all equal to 1.0. E will be set to 1.15 reflecting an average, large project. The estimated effort is $PM_{NS} = 2.94(100)^{1.15} = 586.61$.

Continuing the example, the duration is estimated as $TDEV_{NS} = 3.67(586.6)^{(0.28+0.2\times(1.15-0.91))} = 3.67(586.6)^{0.328} = 29.7$ months. The average number of staff required for the nominal-schedule development is $PM_{NS} / TDEV_{NS} = 586.6 / 29.7 = 19.75$ or about 20 people. In this example, an average 100 KSLOC software project will take about 30 months to complete with an average of 20 people.

2. Sizing

A good size estimate is very important for a good model estimation. However, determining size can be challenging. Projects are generally composed of new code, code reused from other sources--with or without modifications--and automatically translated code. COCOMO II only uses size data that influences effort which is new code and code that is copied and modified.

For new and reused code, a method is used to make them equivalent so they can be rolled up into an aggregate size estimate. The baseline size in COCOMO II is a count of new lines of code. The count for code that is copied and then modified has to be adjusted to create a count that is equivalent to new lines of code. The adjustment takes into account the amount of design, code and testing that was changed. It also considers the understandability of the code and the programmer familiarity with the code.

For automatically translated code, a separate translation productivity rate is used to determine effort from the amount of code to be translated.

The following sections discuss sizing new code and reused code.

2.1 Counting Source Lines of Code (SLOC)

There are several sources for estimating new lines of code. The best source is historical data. For instance, there may be data that will convert Function Points, components, or anything available early in the project to estimate lines of code. Lacking historical data, expert opinion can be used to derive estimates of likely, lowest-likely, and highest-likely size.

Code size is expressed in thousands of source lines of code (KSLOC). A source line of code is generally meant to exclude non-delivered support software such as test drivers. However, if these are developed with the same care as delivered software, with their own reviews, test plans, documentation, etc., then they should be counted [Boehm 1981, pp. 58-59]. The goal is to measure the amount of intellectual work put into program development.

Defining a line of code is difficult because of conceptual differences involved in accounting for executable statements and data declarations in different languages. Difficulties arise when trying to define consistent measures across different programming languages. In COCOMO II, the logical source statement has been chosen as the standard line of code. The Software Engineering Institute (SEI) definition checklist for a logical source statement is used in defining the line of code measure. The SEI has developed this checklist as part of a system of definition checklists, report forms and supplemental forms to support measurement definitions [Park 1992; Goethert et al. 1992].

A SLOC definition checklist is used to support the development of the COCOMO II model. The full checklist is provided at the end of this manual, Table 64. Each checkmark in the "Includes" column identifies a particular statement type or attribute included in the definition, and vice versa for the excludes. Other sections in the definition clarify statement attributes for usage, delivery, functionality, replications and development status.

Some changes were made to the line-of-code definition that departs from the default definition provided in [Park 1992]. These changes eliminate categories of software, which are generally small sources of project effort. For example, not included in the definition are commercial-off-the-shelf software (COTS), government-furnished software (GFS), other products, language support libraries and operating systems, or other commercial libraries. Code generated with source code generators is handled by counting separate operator directives as lines of source code. It is admittedly difficult to count "directives" in a highly visual programming system. As this approach becomes better understood, we hope to provide more specific counting rules. For general source code sizing approaches, such as PERT sizing, expert consensus, analogy, top-down, and bottom-up, see Section 21.4 and Chapter 22 of [Boehm 1981].

2.2 Counting Unadjusted Function Points (UFP)

The function point cost estimation approach is based on the amount of functionality in a software project and a set of individual project factors [Behrens 1983; Kunkler 1985; IFPUG 1994]. Function points are useful estimators since they are based on information that is available early in the project life-cycle. A brief summary of function points and their calculation in support of COCOMO II follows.

Function points measure a software project by quantifying the information processing functionality associated with major external data or control input, output, or file types. Five user function types should be identified as defined in Table 1.

Table 1. User Function Types

Function Point	Description
External Input (EI)	Count each unique user data or user control input type that enters the external boundary of the software system being measured.
External Output (EO)	Count each unique user data or control output type that leaves the external boundary of the software system being measured.
Internal Logical File (ILF)	Count each major logical group of user data or control information in the software system as a logical internal file type. Include each logical file (e.g., each logical group of data) that is generated, used, or maintained by the software system.
External Interface Files (EIF)	Files passed or shared between software systems should be counted as external interface file types within each system.
External Inquiry (EQ)	Count each unique input-output combination, where input causes and generates an immediate output, as an external inquiry type.

Each instance of these function types is then classified by complexity level. The complexity levels determine a set of weights, which are applied to their corresponding function counts to determine the Unadjusted Function Points quantity. This is the Function Point sizing metric used by COCOMO II. The usual Function Point procedure, which is not followed by COCOMO II, involves assessing the degree of influence (DI) of fourteen application characteristics on the software project determined according to a rating scale of 0.0 to 0.05 for each characteristic. The 14 ratings are added together and then added to a base level of 0.65 to produce a general characteristic adjustment factor that ranges from 0.65 to 1.35.

Each of these fourteen characteristics, such as distributed functions, performance, and reusability, thus have a maximum of 5% contribution to estimated effort. Having, for example, a

5% limit on the effect of reuse is inconsistent with COCOMO experience; thus COCOMO II uses Unadjusted Function Points for sizing, and applies its reuse factors, cost drivers, and scale factors to this sizing quantity to account for the effects of reuse, distribution, etc. on project effort.

The COCOMO II procedure for determining Unadjusted Function Points follows the definitions in [IFPUG 1994]. This four step procedure, which follows, is used in both the Early Design and the Post-Architecture models.

1. Determine function counts by type. The unadjusted function counts should be counted by a lead technical person based on information in the software requirements and design documents. The number of each of the five user function types should be counted (Internal Logical File (ILF), External Interface File (EIF), External Input (EI), External Output (EO), and External Inquiry (EQ)). See [IFPUG 1994] for more detailed interpretations of the counting rules for those quantities.
2. Determine complexity levels. Classify each function count into Low, Average and High complexity levels depending on the number of data element types contained and the number of file types referenced. Use the scheme in Table 2.

Table 2. FP Counting Weights

For Internal Logical Files and External Interface Files

	Data Elements		
Record Elements	**1 - 19**	**20 - 50**	**51+**
1	Low	Low	Avg.
2 - 5	Low	Avg.	High
6+	Avg.	High	High

For External Output and External Inquiry

	Data Elements		
File Types	**1 - 5**	**6 - 19**	**20+**
0 or 1	Low	Low	Avg.
2 - 3	Low	Avg.	High
4+	Avg.	High	High

For External Input

	Data Elements		
File Types	**1 - 4**	**5 - 15**	**16+**
0 or 1	Low	Low	Avg.
2 - 3	Low	Avg.	High
3+	Avg.	High	High

3. Apply complexity weights. Weight the number of function types at each complexity level using the following scheme (the weights reflect the relative effort required to implement the function):

Table 3. UFP Complexity Weights

Function Type	Complexity-Weight Low	Average	High
Internal Logical Files	7	10	15
External Interfaces Files	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

4. Compute Unadjusted Function Points. Add all the weighted functions counts to get one number, the Unadjusted Function Points.

2.3 Relating UFPs to SLOC

Next, convert the Unadjusted Function Points (UFP) to Lines of Code. The unadjusted function points have to be converted to source lines of code in the implementation language (Ada, C, C++, Pascal, etc.). COCOMO II does this for both the Early Design and Post-Architecture models by using backfiring tables to convert Unadjusted Function Points into equivalent SLOC. The current conversion ratios shown in Table 4 are from [Jones 1996]. Updates to these conversion ratios as well as additional ratios can be found at http://www.spr.com/library/0Langtbl.htm.

Table 4. UFP to SLOC Conversion Ratios

Language	Default SLOC / UFP	Language	Default SLOC / UFP
Access	38	Jovial	107
Ada 83	71	Lisp	64
Ada 95	49	Machine Code	640
AI Shell	49	Modula 2	80
APL	32	Pascal	91
Assembly - Basic	320	PERL	27
Assembly - Macro	213	PowerBuilder	16
Basic - ANSI	64	Prolog	64
Basic - Compiled	91	Query – Default	13
Basic - Visual	32	Report Generator	80
C	128	Second Generation Language	107
C++	55	Simulation – Default	46
Cobol (ANSI 85)	91	Spreadsheet	6
Database – Default	40	Third Generation Language	80
Fifth Generation Language	4	Unix Shell Scripts	107
First Generation Language	320	USR_1	1
Forth	64	USR_2	1
Fortran 77	107	USR_3	1
Fortran 95	71	USR_4	1
Fourth Generation Language	20	USR_5	1
High Level Language	64	Visual Basic 5.0	29
HTML 3.0	15	Visual C++	34
Java	53		

USR_1 through USR_5 are five extra slots provided by USC COCOMO II.2000 to accommodate user-specified additional implementation languages. These ratios are easy to determine with historical data or with a recently completed project. It would be prudent to determine your own ratios for your local environment.

2.4 Aggregating New, Adapted, and Reused Code

A product's size discussed thus far has been for new development. Code that is taken from another source and used in the product under development also contributes to the product's effective size. Preexisting code that is treated as a black-box and plugged into the product is called reused code. Preexisting code that is treated as a white-box and is modified for use with the product is called adapted code. The effective size of reused and adapted code is adjusted to be its equivalent in new code. The adjusted code is called equivalent source lines of code (ESLOC). The adjustment is based on the additional effort it takes to modify the code for inclusion in the product. The sizing model treats reuse with function points and source lines of code the same in either the Early Design model or the Post-Architecture model.

2.4.1 Nonlinear Reuse Effects

Analysis in [Selby 1988] of reuse costs across nearly three thousand reused modules in the NASA Software Engineering Laboratory indicates that the reuse cost function, relating the amount of modification of the reused code to the resulting cost to reuse, is nonlinear in two significant ways (see Figure 1). The effort required to reuse code does not start at zero. There is generally a cost of about 5% for assessing, selecting, and assimilating the reusable component.

Figure 1 shows the results of the NASA analysis as blocks of relative cost. A dotted line is superimposed on the blocks of relative cost to show increasing cost as more of the reused code is modified. (The solid lines are labeled AAM for Adaptation Adjustment Modifier. AAM is explained in Equation 4.) It can be seen that small modifications in the reused product generate disproportionately large costs. This is primarily because of two factors: the cost of understanding the software to be modified, and the relative cost of checking module interfaces.

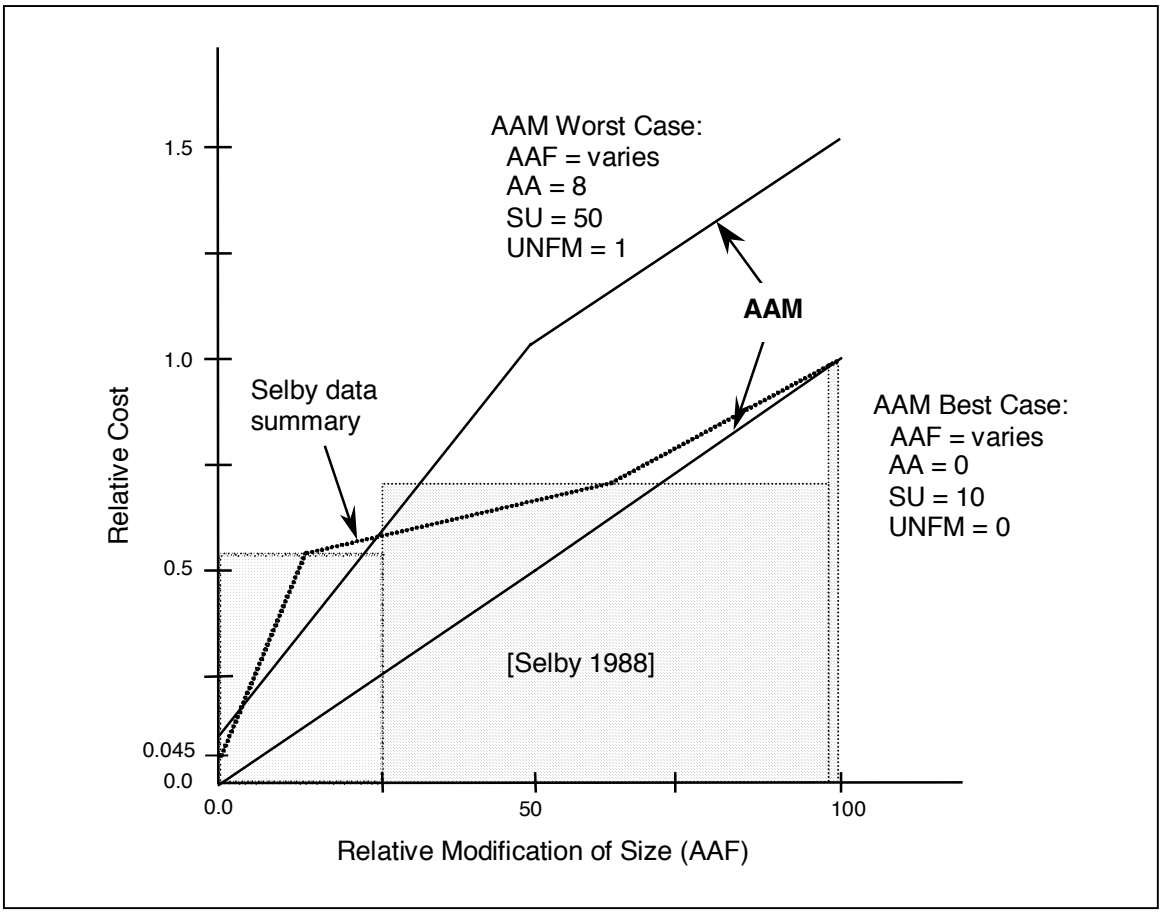


Figure 1. Non-Linear Reuse Effects

[Parikh-Zvegintzov 1983] contains data indicating that 47% of the effort in software maintenance involves understanding the software to be modified. Thus, as soon as one goes from unmodified (black-box) reuse to modified-software (white-box) reuse, one encounters this software understanding penalty. Also, [Gerlich-Denskat 1994] shows that, if one modifies k out of m software modules, the number of module interface checks required, N, is expressed in Equation 3.

$$N = k \times (m - k) + k \times \left(\frac{k - 1}{2} \right) \qquad \textbf{Eq. 3}$$

Figure 2 shows this relation between the number of modules modified k and the resulting number, N, of module interface checks required for an example of m = 10 modules. In this example, modifying 20% (2 of 10) of the modules required revalidation of 38% (17 of 45) of the interfaces.

The shape of this curve is similar for other values of m. It indicates that there are nonlinear effects involved in the module interface checking which occurs during the design, code, integration, and test of modified software.

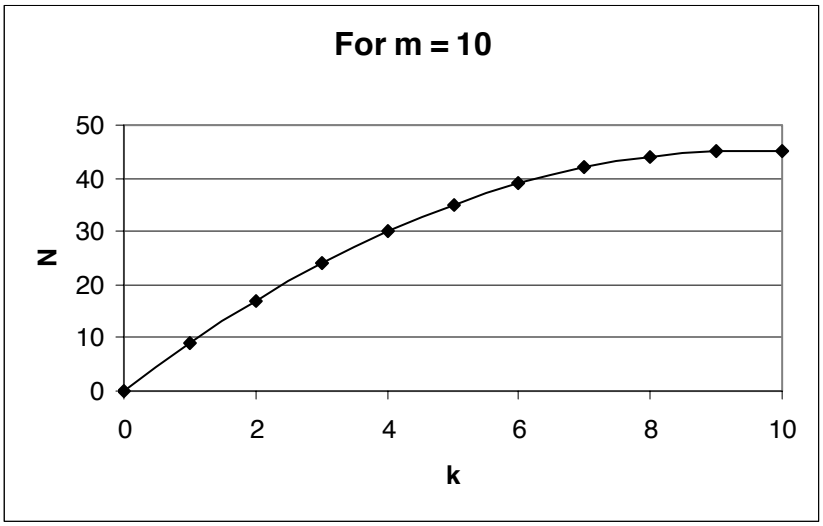


Figure 2. Number of Module Interface Checks, N, vs. Modules Modified, k

The size of both the software understanding penalty and the module interface-checking penalty can be reduced by good software structuring. Modular, hierarchical structuring can reduce the number of interfaces which need checking [Gerlich-Denskat 1994], and software that is well-structured, explained, and related to its mission will be easier to understand. COCOMO II reflects this in its allocation of estimated effort for modifying reusable software.

2.4.2 A Reuse Model

The COCOMO II treatment of software reuse uses a nonlinear estimation model, Equation 4. This involves estimating the amount of software to be adapted and three degree-of-modification factors: the percentage of design modified (DM), the percentage of code modified (CM), and the percentage of integration effort required for integrating the adapted or reused software (IM). These three factors use the same linear model as used in COCOMO 81, but COCOMO II adds some nonlinear increments to the relation of Adapted KSLOC of Equivalent KSLOC to reflect the non-linear tendencies of the model. These are explained next.

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

$$\text{where AAM} = \begin{cases} \dfrac{[\text{AA} + \text{AAF}(1 + (0.02 \times \text{SU} \times \text{UNFM}))]}{100}, & \text{for AAF} \le 50 \\ \dfrac{[\text{AA} + \text{AAF} + (\text{SU} \times \text{UNFM})]}{100}, & \text{for AAF} > 50 \end{cases} \qquad \textbf{Eq. 4}$$

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

The Software Understanding increment (SU) is obtained from Table 5. SU is expressed quantitatively as a percentage. If the software is rated very high on structure, applications clarity, and self-descriptiveness, the software understanding and interface-checking penalty is

10%. If the software is rated very low on these factors, the penalty is 50%. SU is determined by taking the subjective average of the three categories.

Table 5. Rating Scale for Software Understanding Increment SU

	Very Low	Low	Nominal	High	Very High
Structure	Very low cohesion, high coupling, spaghetti code.	Moderately low cohesion, high coupling.	Reasonably well-structured; some weak areas.	High cohesion, low coupling.	Strong modularity, information hiding in data / control structures.
Application Clarity	No match between program and application world-views.	Some correlation between program and application.	Moderate correlation between program and application.	Good correlation between program and application.	Clear match between program and application world-views.
Self-Descriptive-ness	Obscure code; documentation missing, obscure or obsolete.	Some code commentary and headers; some useful documentation.	Moderate level of code commentary, headers, documentation.	Good code commentary and headers; useful documentation; some weak areas.	Self-descriptive code; documentation up-to-date, well-organized, with design rationale.
SU Increment to ESLOC	50	40	30	20	10

The other nonlinear reuse increment deals with the degree of Assessment and Assimilation (AA) needed to determine whether a reused software module is appropriate to the application, and to integrate its description into the overall product description. Table 6 provides the rating scale and values for the assessment and assimilation increment. AA is a percentage.

Table 6. Rating Scale for Assessment and Assimilation Increment (AA)

AA Increment	Level of AA Effort
0	None
2	Basic module search and documentation
4	Some module Test and Evaluation (T&E), documentation
6	Considerable module T&E, documentation
8	Extensive module T&E, documentation

The amount of effort required to modify existing software is a function not only of the amount of modification (AAF) and understandability of the existing software (SU), but also of the programmer's relative unfamiliarity with the software (UNFM). The UNFM factor is applied multiplicatively to the software understanding effort increment. If the programmer works with the software every day, the 0.0 multiplier for UNFM will add no software understanding increment. If the programmer has never seen the software before, the 1.0 multiplier will add the full software understanding effort increment. The rating of UNFM is shown in Table 7.

Table 7. Rating Scale for Programmer Unfamiliarity (UNFM)

UNFM Increment	Level of Unfamiliarity
0.0	Completely familiar
0.2	Mostly familiar
0.4	Somewhat familiar
0.6	Considerably familiar
0.8	Mostly unfamiliar
1.0	Completely unfamiliar

Equation 4 is used to determine an equivalent number of new source lines of code. The calculation of equivalent SLOC is based on the product size being adapted and a modifier that accounts for the effort involved in fitting adapted code into an existing product, called Adaptation Adjustment Modifier (AAM). The term (1 – AT/100) is for automatically translated code and is discussed in Section 2.2.6.

AAM uses the factors discussed above, Software Understanding (SU), Programmer Unfamiliarity (UNFM), and Assessment and Assimilation (AA) with a factor called the Adaptation Adjustment Factor (AAF). AAF contains the quantities DM, CM, and IM where:

- DM (Percent Design Modified) is the percentage of the adapted software's design which is modified in order to adapt it to the new objectives and environment. (This is necessarily a subjective quantity.)
- CM (Percent Code Modified) is the percentage of the adapted software's code which is modified in order to adapt it to the new objectives and environment.
- IM (Percent of Integration Required for Adapted Software) is the percentage of effort required to integrate the adapted software into an overall product and to test the resulting product as compared to the normal amount of integration and test effort for software of comparable size.

If there is no DM or CM (the component is being used unmodified) then there is no need for SU. If the code is being modified then SU applies.

The range of AAM is shown in Figure 1. Under the worst case, it can take twice the effort to modify a reused module than it takes to develop it as new (the value of AAM can exceed 100). The best case follows a one for one correspondence between adapting an existing product and developing it from scratch.

2.4.3 Guidelines for Quantifying Adapted Software

This section provides guidelines to estimate adapted software factors for different categories of code using COCOMO II. The *New* category refers to software developed from scratch. *Adapted* code is preexisting code that has some changes to it, while *reused* code has no changes to the preexisting source (i.e. used as-is). *COTS* is off-the-shelf software that is generally treated the same as reused code when there are no changes to it. One difference is that there may be some new glue code associated with it that also needs to be counted (this may happen with reused software, but here the option of modifying the source code may make adapting the software more attractive).

Since there is no source code modified in reused and COTS, DM=0, CM=0, and SU and UNFM don't apply. AA and IM can have non-zero values in this case. Reuse doesn't mean free integration and test. However in the reuse approach, with well-architected product-lines, the integration and test is minimal.

For adapted software, CM > 0, DM is usually > 0, and all other reuse factors normally have non-zero values. IM is expected to be at least moderate for adapted software, but can be higher than 100% for adaptation into more complex applications. Table 8 shows the valid ranges of reuse factors with additional notes for the different categories.

Table 8. Adapted Software Parameter Constraints and Guidelines

Code Category	Reuse Parameters					
	DM	CM	IM	AA	SU	UNFM
New all original software			not applicable			
Adapted changes to preexisting software	0% - 100% normally > 0%	0^{+}% - 100% usually > DM and must be > 0%	0% - 100+% IM usually moderate and can be > 100%	0% – 8%	0% - 50%	0 - 1
Reused unchanged existing software	0%	0%	0% - 100% rarely 0%, but could be very small	0% – 8%	not applicable	
COTS off-the-shelf software (often requires new glue code as a wrapper around the COTS)	0%	0%	0% - 100%	0% – 8%	not applicable	

2.5 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 caused 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. For example, a project which delivers 100,000 instructions but discards the equivalent of an additional 20,000 instructions has an REVL value of 20. This would be used to adjust the project's effective size to 120,000 instructions for a COCOMO II estimation.

The use of REVL for computing size in given in Equation 5.

$$\text{Size} = \left(1 + \frac{\text{REVL}}{100}\right) \times \text{Size}_{\text{D}} \qquad \textbf{Eq. 5}$$

where Size_{D} is the reuse - equivalent of the delivered software.

2.6 Automatically Translated Code

The COCOMO II reuse model needs additional refinement to estimate the costs of software reengineering and conversion. The major difference in reengineering and conversion is the efficiency of automated tools for software restructuring. These can lead to very high values for the percentage of code modified (CM in the COCOMO II reuse model), but with very little corresponding effort. For example, in the NIST reengineering case study [Ruhl-Gunn 1991], 80% of the code (13,131 COBOL source statements) was re-engineered by automatic translation, and the actual reengineering effort, 35 Person-Months, was more than a factor of 4 lower than the COCOMO estimate of 152 person months.

The COCOMO II reengineering and conversion estimation approach involves estimating an additional factor, AT, the percentage of the code that is re-engineered by automatic translation. Based on an analysis of the project data above, the default productivity value for automated translation is 2400 source statements per person month. This value could vary with different technologies and is designated in the COCOMO II model as another factor called ATPROD. In the NIST case study ATPROD = 2400. Equation 6 shows how automated translation affects the estimated effort, PM_{Auto}.

$$PM_{Auto} = \frac{\text{Adapted SLOC} \times \left(AT/100\right)}{\text{ATPROD}} \qquad \textbf{Eq. 6}$$

The NIST case study also provides useful guidance on estimating the AT factor, which 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. The NIST data on percentage of automated translation (from an original batch processing application without COTS utilities) are given in Table 9 [Ruhl-Gunn 1991].

Table 9. Variation in Percentage of Automated Re-engineering

Re-engineering Target	AT (% automated translation)
Batch processing	96%
Batch with SORT	90%
Batch with DBMS	88%
Batch, SORT, DBMS	82%
Interactive	50%

Automated translation is considered to be a separate activity from development. Thus, its Adapted SLOC are not included as Size in Equivalent KSLOC, and its PM_{AUTO} are not included in PM_{NS} in estimating the project's schedule. If the automatically translated Adapted SLOC count is included as Size in the Equivalent KSLOC, it must be backed out to prevent double counting. This is done by adding the term (1 – AT/100) to the equation for Equivalent KSLOC, Equation 2.4.

2.7 Sizing Software Maintenance

COCOMO II differs from COCOMO 81 in applying the COCOMO II scale factors to the size of the modified code rather than applying the COCOMO 81 modes to the size of the product being modified. Applying the scale factors to a 10 million SLOC product produced overlarge

estimates as most of the product was not being touched by the changes. COCOMO II accounts for the effects of the product being modified via its software understanding and unfamiliarity factors discussed for reuse in Section 2.4.2.

The scope of "software maintenance" follows the COCOMO 81 guidelines in [Boehm 1981; pp.534-536]. It includes adding new capabilities and fixing or adapting existing capabilities. It excludes major product rebuilds changing over 50% of the existing software, and development of sizable (over 20% changed) interfacing systems requiring little rework of the existing system.

The maintenance size is normally obtained via Equation 7, when the base code size is known and the percentage of change to the base code is known.

$$(\text{Size})_M = \left[(\text{Base Code Size}) \times \text{MCF}\right] \times \text{MAF} \qquad \textbf{Eq. 7}$$

The Maintenance Adjustment Factor (MAF) is discussed below. But first, the percentage of change to the base code is called the Maintenance Change Factor (MCF). The MCF is similar to the Annual Change Traffic in COCOMO 81, except that maintenance periods other than a year can be used. Conceptually the MCF represents the ratio in Equation 8:

$$\text{MCF} = \frac{\text{Size Added} + \text{Size Modified}}{\text{Base Code Size}} \qquad \textbf{Eq. 8}$$

A simpler version can be used when the fraction of code added or modified to the existing base code during the maintenance period is known. Deleted code is not counted.

$$(\text{Size})_M = (\text{Size Added} + \text{Size Modified}) \times \text{MAF} \qquad \textbf{Eq. 9}$$

The size can refer to thousands of source lines of code (KSLOC), Function Points, or Application Points. When using Function Points or Application Points, it is better to estimate MCF in terms of the fraction of the overall application being changed, rather than the fraction of inputs, outputs, screens, reports, etc. touched by the changes. Our experience indicates that counting the items touched can lead to significant over estimates, as relatively small changes can touch a relatively large number of items. In some very large COBOL programs, we found ratios of 2 to 3 FP-touched/SLOC-changed as compared to 91 FP/SLOC for development.

The Maintenance Adjustment Factor (MAF), Equation 10, is used to adjust the effective maintenance size to account for software understanding and unfamiliarity effects, as with reuse. COCOMO II uses the Software Understanding (SU) and Programmer Unfamiliarity (UNFM) factors from its reuse model (discussed in Section 2.4.2) to model the effects of well or poorly structured/understandable software on maintenance effort.

$$\text{MAF} = 1 + \left(\frac{\text{SU}}{100} \times \text{UNFM}\right) \qquad \textbf{Eq. 10}$$

The use of $(\text{Size})_M$ in determining maintenance effort, Equation 9, is discussed in Section 5.

3. Effort Estimation

In COCOMO II effort is expressed as Person-Months (PM). A person month is the amount of time one person spends working on the software development project for one month. COCOMO II treats the number of person-hours per person-month, PH/PM, as an adjustable factor with a nominal value of 152 hours per Person-Month. This number excludes time typically devoted to holidays, vacations, and weekend time off. The number of person-months is different from the time it will take the project to complete; this is called the development schedule or Time to Develop, TDEV. For example, a project may be estimated to require 50 PM of effort but have a schedule of 11 months. If you use a different value of PH/PM–say, 160 instead of 152–COCOMO II adjusts the PM estimate accordingly (in this case, reducing by about 5%). This reduced PM will result in a smaller estimate of development schedule.

The COCOMO II effort estimation model was introduced in Equation 1, and is summarized in Equation 11. This model form is used for both the Early Design and Post-Architecture cost models to estimate effort between the end points of LCO and IOC for the MBASE/RUP and SRR and SAR for the Waterfall lifecycle models (see Section 6.2). The inputs are the Size of software development, a constant, A, an exponent, E, and a number of values called effort multipliers (EM). The number of effort multipliers depends on the model.

$$PM = A \times Size^{E} \times \prod_{i=1}^{n} EM_i \qquad \textbf{Eq. 11}$$

where A = 2.94 (for COCOMO II.2000)

The exponent E is explained in detail in Section 3.1. The effort multipliers are explained in Section 3.2. The constant, A, approximates a productivity constant in PM/KSLOC for the case where E = 1.0. Productivity changes as E increases because of the non-linear effects on Size. The constant A is initially set when the model is calibrated to the project database reflecting a global productivity average. The COCOMO model should be calibrated to local data which then reflects the local productivity and improves the model's accuracy. Section 7 discusses how to calibrate the model to the local environment.

The Size is KSLOC. This is derived from estimating the size of software modules that will constitute the application program. It can also be estimated from unadjusted function points (UFP), converted to SLOC, then divided by one thousand. Procedures for counting SLOC or UFP were explained in Section 2, including adjustments for reuse, requirements evolution, and automatically translated code.

Cost drivers are used to capture characteristics of the software development that affect the effort to complete the project. A cost driver is a model factor that "drives" the cost (in this case Person-Months) estimated by the model. All COCOMO II cost drivers have qualitative rating levels that express the impact of the driver on development effort. These ratings can range from Extra Low to Extra High. Each rating level of every multiplicative cost driver has a value, called an *effort multiplier* (EM), associated with it. This scheme translates a cost driver's qualitative rating into a quantitative one for use in the model. The EM value assigned to a multiplicative cost driver's nominal rating is 1.00. If a multiplicative cost driver's rating level

causes more software development effort, then its corresponding EM is above 1.0. Conversely, if the rating level reduces the effort then the corresponding EM is less than 1.0.

The rating of cost drivers is based on a strong rationale that they would independently explain a significant source of project effort or productivity variation. The difference between the Early Design and Post-Architecture models are the number of multiplicative cost drivers and the areas of influence they explain. There are seven multiplicative cost drivers for the Early Design model and seventeen multiplicative cost drivers for the Post-Architecture model. Each set is explained with its model later in the manual.

It turns out that the 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. These are discussed next.

What is not apparent in the model definition form given in Equation 11 is that there are some model drivers that apply only to the project as a whole. The scale factors in the exponent, E, are only used at the project level. Additionally, one of the multiplicative cost drivers that is in the product of effort multipliers, Required Development Schedule (SCED) is only used at the project level. The other multiplicative cost drivers, which are all represented in the product of effort multipliers, and size apply to individual project components. The model can be used to estimate effort for a project that has only one component or multiple components. For multi-component projects the project-level cost drivers apply to all components, see Section 3.3.

3.1 Scale Factors

The exponent E in Equation 11 is an aggregation of five *scale factors* (SF) that account for the relative economies or diseconomies of scale encountered for software projects of different sizes [Banker et al. 1994]. If $E < 1.0$, the project exhibits economies of scale. If the product's size is doubled, the project effort is less than doubled. The project's productivity increases as the product size is increased. Some project economies of scale can be achieved via project-specific tools (e.g., simulations, testbeds), but in general these are difficult to achieve. For small projects, fixed start-up costs such as tool tailoring and setup of standards and administrative reports are often a source of economies of scale.

If $E = 1.0$, the economies and diseconomies of scale are in balance. This linear model is often used for cost estimation of small projects.

If $E > 1.0$, the project exhibits diseconomies of scale. This is generally because of two main factors: growth of interpersonal communications overhead and growth of large-system integration overhead. Larger projects will have more personnel, and thus more interpersonal communications paths consuming overhead. Integrating a small product as part of a larger product requires not only the effort to develop the small product, but also the additional overhead effort to design, maintain, integrate, and test its interfaces with the remainder of the product. See [Banker et al. 1994] for a further discussion of software economies and diseconomies of scale.

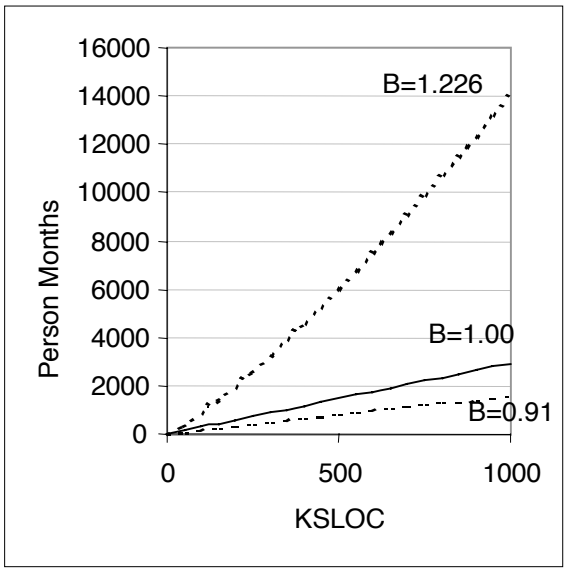


Figure 3. Diseconomies of Scale Effect on Effort

Equation 12 defines the exponent, E, used in Equation 11. Table 10 provides the rating levels for the COCOMO II scale factors. The selection of scale factors is based on the rationale that they are a significant source of exponential variation on a project's effort or productivity variation. Each scale factors has a range of rating levels, from Very Low to Extra High. Each rating level has a weight. The specific value of the weight is called a *scale factor* (SF). The project's scale factors, the selected scale factors ratings, are summed and used to determine a scale exponent, E, via Equation 12. The B term in the equation is a constant that can be calibrated [Boehm et al. 2000].

$$E = B + 0.01 \times \sum_{j=1}^{5} SF_j \qquad \textbf{Eq. 12}$$

where B = 0.91 (for COCOMO II.2000)

For example, scale factors in COCOMO II with an Extra High rating are each assigned a scale factor weight of (0). Thus, a 100 KSLOC project with Extra High ratings for all scale factors will have $\Sigma SF_j = 0$, E = 0.91, and a relative effort of $2.94(100)^{0.91} = 194$ PM. For the COCOMO II.2000 calibration of scale factors in Table 10, a project with Very Low ratings for all scale factors will have $\Sigma SF_j = 31.6$, E = 1.226, and a relative effort of $2.94(100)^{1.226} = 832$ PM. This represents a large variation, but the increase involved in a one-unit rating level change in one of the scale factors is only about 6%. For very large (1,000 KSLOC) products, the effect of the scale factors is much larger, as seen in Figure 3.

Table 10. Scale Factor Values, SF_j, for COCOMO II Models

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	thoroughly familiar
SF_j:	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	general goals
SF_j:	5.07	4.05	3.04	2.03	1.01	0.00
RESL	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
SF_j:	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions
SF_j:	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	The estimated Equivalent Process Maturity Level (EPML) or					
	SW-CMM Level 1 Lower	SW-CMM Level 1 Upper	SW-CMM Level 2	SW-CMM Level 3	SW-CMM Level 4	SW-CMM Level 5
SF_j:	7.80	6.24	4.68	3.12	1.56	0.00

The two scale factors, Precedentedness and Flexibility largely capture the differences between the Organic, Semidetached, and Embedded modes of the original COCOMO model [Boehm 1981]. Table 11 and Table 12 reorganize [Boehm 1981; Table 6.3] to map its project features onto the Precedentedness and Development Flexibility scales. These tables can be used as a more in depth explanation for the PREC and FLEX rating scales given in Table 10.

3.1.1 Precedentedness (PREC)

If a product is similar to several previously developed projects, then the precedentedness is high.

Table 11. Precedentedness Rating Levels

Feature	Very Low	Nominal / High	Extra High
Organizational understanding of product objectives	General	Considerable	Thorough
Experience in working with related software systems	Moderate	Considerable	Extensive
Concurrent development of associated new hardware and operational procedures	Extensive	Moderate	Some

Table 11. Precedentedness Rating Levels

Feature	Very Low	Nominal / High	Extra High
Need for innovative data processing architectures, algorithms	Considerable	Some	Minimal

3.1.2 Development Flexibility (FLEX)

Table 12. Development Flexibility Rating Levels

Feature	Very Low	Nominal / High	Extra High
Need for software conformance with pre-established requirements	Full	Considerable	Basic
Need for software conformance with external interface specifications	Full	Considerable	Basic
Combination of inflexibilities above with premium on early completion	High	Medium	Low

The PREC and FLEX scale factors are largely intrinsic to a project and uncontrollable. The next three scale factors identify management controllables by which projects can reduce diseconomies of scale by reducing sources of project turbulence, entropy, and rework.

3.1.3 Architecture / Risk Resolution (RESL)

This factor combines two of the scale factors in Ada COCOMO, "Design Thoroughness by Product Design Review (PDR)" and "Risk Elimination by PDR" [Boehm-Royce 1989; Figures 4 and 5]. Table 13 consolidates the Ada COCOMO ratings to form a more comprehensive definition for the COCOMO II RESL rating levels. It also relates the rating level to the MBASE/RUP Life Cycle Architecture (LCA) milestone as well as to the waterfall PDR milestone. The RESL rating is the subjective weighted average of the listed characteristics.

Table 13. RESL Rating Levels

Characteristic	Very Low	Low	Nominal	High	Very High	Extra High
Risk Management Plan identifies all critical risk items, establishes milestones for resolving them by PDR or LCA.	None	Little	Some	Generally	Mostly	Fully
Schedule, budget, and internal milestones through PDR or LCA compatible with Risk Management Plan.	None	Little	Some	Generally	Mostly	Fully
Percent of development schedule devoted to establishing architecture, given general product objectives.	5	10	17	25	33	40

Table 13. RESL Rating Levels

Characteristic	Very Low	Low	Nominal	High	Very High	Extra High
Percent of required top software architects available to project.	20	40	60	80	100	120
Tool support available for resolving risk items, developing and verifying architectural specs.	None	Little	Some	Good	Strong	Full
Level of uncertainty in key architecture drivers: mission, user interface, COTS, hardware, technology, performance.	Extreme	Significant	Consider-able	Some	Little	Very Little
Number and criticality of risk items.	> 10 Critical	5-10 Critical	2-4 Critical	1 Critical	> 5Non-Critical	< 5 Non-Critical

3.1.4 Team Cohesion (TEAM)

The Team Cohesion scale factor accounts for the sources of project turbulence and entropy because of difficulties in synchronizing the project's stakeholders: users, customers, developers, maintainers, interfacers, others. These difficulties may arise from differences in stakeholder objectives and cultures; difficulties in reconciling objectives; and stakeholders' lack of experience and familiarity in operating as a team. Table 14 provides a detailed definition for the overall TEAM rating levels. The final rating is the subjective weighted average of the listed characteristics.

Table 14. TEAM Rating Components

Characteristic	Very Low	Low	Nominal	High	Very High	Extra High
Consistency of stakeholder objectives and cultures	Little	Some	Basic	Consider-able	Strong	Full
Ability, willingness of stakeholders to accommodate other stakeholders' objectives	Little	Some	Basic	Consider-able	Strong	Full
Experience of stakeholders in operating as a team	None	Little	Little	Basic	Consider-able	Extensive
Stakeholder teambuilding to achieve shared vision and commitments	None	Little	Little	Basic	Consider-able	Extensive

3.1.5 Process Maturity (PMAT)

Overall Maturity Levels

The procedure for determining PMAT is organized around the Software Engineering Institute's Capability Maturity Model (CMM). The time period for rating Process Maturity is the time the project starts. There are two ways of rating Process Maturity. The first captures the result of an organized evaluation based on the CMM, and is explained in Table 15.

Table 15. PMAT Ratings for Estimated Process Maturity Level (EPML)

PMAT Rating	Maturity Level	EPML
Very Low	CMM Level 1 (lower half)	0
Low	CMM Level 1 (upper half)	1
Nominal	CMM Level 2	2
High	CMM Level 3	3
Very High	CMM Level 4	4
Extra High	CMM Level 5	5

Key Process Area Questionnaire

The second is organized around the 18 Key Process Areas (KPAs) in the SEI Capability Maturity Model [Paulk et al. 1995]. The procedure for determining PMAT is to decide the percentage of compliance for each of the KPAs. If the project has undergone a recent CMM Assessment, then the percentage compliance for the overall KPA (based on KPA Key Practice compliance assessment data) is used. If an assessment has not been done, then the levels of compliance to the KPA's goals are used (with the Likert scale in Table 16) to set the level of compliance. The goal-based level of compliance is determined by a judgment-based averaging across the goals for each Key Process Area. See [Paulk et al. 1995] for more information on the KPA definitions, goals and activities.

Table 16. KPA Rating Levels

Key Process Areas (KPA)	Almost Always[1]	Frequently[2]	About Half[3]	Occasionally[4]	Rarely if Ever[5]	Does Not Apply[6]	Don't Know[7]
Requirements Management • System requirements allocated to software are controlled to establish a baseline for software engineering and management use. • Software plans, products, and activities are kept consistent with the system requirements allocated to software.	•	•	•	•	•	•	•
Software Project Planning • Software estimates are documented for use in planning and tracking the software project. • Software project activities and commitments are planned and documented. • Affected groups and individuals agree to their commitments related to the software project.	•	•	•	•	•	•	•

Table 16. KPA Rating Levels

Key Process Areas (KPA)	Almost Always[1]	Frequently[2]	About Half[3]	Occasionally[4]	Rarely if Ever[5]	Does Not Apply[6]	Don't Know[7]
Software Project Tracking and Oversight • Actual results and performances are tracked against the software plans • Corrective actions are taken and managed to closure when actual results and performance deviate significantly from the software plans. • Changes to software commitments are agreed to by the affected groups and individuals.	•	•	•	•	•	•	•
Software Subcontract Management • The prime contractor selects qualified software subcontractors. • The prime contractor and the subcontractor agree to their commitments to each other. • The prime contractor and the subcontractor maintain ongoing communications. • The prime contractor tracks the subcontractor's actual results and performance against its commitments.	•	•	•	•	•	•	•
Software Quality Assurance (SQA) • SQA activities are planned. • Adherence of software products and activities to the applicable standards, procedures, and requirements is verified objectively. • Affected groups and individuals are informed of software quality assurance activities and results. • Noncompliance issues that cannot be resolved within the software project are addressed by senior management.	•	•	•	•	•	•	•
Software Configuration Management (SCM) • SCM activites are planned. • Selected workproducts are identified, controlled, and available. • Changes to identified work products are controlled. • Affected groups and individuals are informed of the status and content of software baselines.	•	•	•	•	•	•	•
Organization Process Focus • Software process development and improvement activities are coordinated across the organization. • The strengths and weaknesses of the software processes used are identified relative to a process standard. • Organization-level process development and improvement activities are planned.	•	•	•	•	•	•	•
Organization Process Definition • A standard software process for the organiation is developed and maintained. • Information related to the use of the organization's standard software process by the software projects is collected, reviewed, and made available.	•	•	•	•	•	•	•

Table 16. KPA Rating Levels

Key Process Areas (KPA)	Almost Always[1]	Frequently[2]	About Half[3]	Occasionally[4]	Rarely if Ever[5]	Does Not Apply[6]	Don't Know[7]
Training Program • Training activities are planned. • Training for developing the skills and knowledge needed to perform software management and technical roles is provided. • Individuals in the software engineering group and software-related groups receive the training necessary to perform their roles.	•	•	•	•	•	•	•
Integrated Software Management • The project's defined software process is a tailored version of the organization's standard software process. • The project is planned and managed according to the project's defined software process.	•	•	•	•	•	•	•
Software Product Engineering • The software engineering tasks are defined, integrated, and consistently performed to produce the software • Software work products are kept consistent with each other.	•	•	•	•	•	•	•
Intergroup Coordination • The customer's requirements are agreed to by all affected groups. • The commitments between the engineering groups are agreed to by the affected groups. • The engineering groups identify, track, and resolve intergroup issues.	•	•	•	•	•	•	•
Peer Reviews • Peer review activities are planned. • Defects in the software work products are identified and removed.	•	•	•	•	•	•	•
Quantitative Process Management • The quantitative process management activities are planned. • The process performance of the project's defined software process is controlled quantitatively. • The process capability of the organization's standard software process is known in quantitative terms.	•	•	•	•	•	•	•
Software Quality Management • The project's software quality management activities are planned. • Measurable goals of software product quality and their priorities are defined. • Actual progress toward achieving the quality goals for the software products is quantified and managed.	•	•	•	•	•	•	•
Defect Prevention • Defect prevention activities are planned. • Common causes of defects are sought out and identified. • Common causes of defects are priortized and systematically eliminated.	•	•	•	•	•	•	•

Table 16. KPA Rating Levels

Key Process Areas (KPA)	Almost Always[1]	Frequently[2]	About Half[3]	Occasionally[4]	Rarely if Ever[5]	Does Not Apply[6]	Don't Know[7]
Technology Change Management • Incorporation of technology changes are planned. • New technologies are evaluated to determine their effect on quality and productivity. • Appropriate new technologies are transferred into normal practice across the organization.	•	•	•	•	•	•	•
Process Change Management • Continuous process improvement is planned. • Participation in the organization's software process improvement activities is organization wide. • The organization's standard software process and the project's defined software processes are improved continuously.	•	•	•	•	•	•	•

1. Check **Almost Always** when the goals are consistently achieved and are well established in standard operating procedures (over 90% of the time).
2. Check **Frequently** when the goals are achieved relatively often, but sometimes are omitted under difficult circumstances (about 60 to 90% of the time).
3. Check **About Half** when the goals are achieved about half of the time (about 40 to 60% of the time).
4. Check **Occasionally** when the goals are sometimes achieved, but less often (about 10 to 40% of the time).
5. Check **Rarely If Ever** when the goals are rarely if ever achieved (less than 10% of the time).
6. Check **Does Not Apply** when you have the required knowledge about your project or organization and the KPA, but you feel the KPA does not apply to your circumstances.
7. Check **Don't Know** when you are uncertain about how to respond for the KPA.

An equivalent process maturity level (EPML) is computed as five times the average compliance level of all n rated KPAs for a single project (Does Not Apply and Don't Know are not counted which sometimes makes n less than 18). After each KPA is rated, the rating level is weighted (100% for Almost Always, 75% for Frequently, 50% for About Half, 25% for Occasionally, 1% for Rarely if Ever). The EPML is calculated as in Equation 2-13.

$$\text{EPML} = 5 \times \left(\sum_{i=1}^{n} \frac{\text{KPA\%}_i}{100} \right) \times \frac{1}{n} \qquad \textbf{Eq. 13}$$

An EPML of 0 corresponds with a PMAT rating level of Very Low in the rating scales of Table 10 and Table 15.

The COCOMO II project is tracking the progress of the recent CMM Integration (CMM-I) activity to determine likely future revisions in the definition of PMAT.

3.2 Effort Multipliers

3.2.1 Post-Architecture Cost Drivers

This model is the most detailed. It is intended to be used when a software life-cycle architecture has been developed. This model is used in the development and maintenance of software products in the Application Generators, System Integration, or Infrastructure sectors [Boehm et al. 2000].

The seventeen Post-Architecture effort multipliers (EM) are used in the COCOMO II model to adjust the nominal effort, Person-Months, to reflect the software product under development, see Equation 11. Each multiplicative cost driver is defined below by a set of rating levels and a corresponding set of effort multipliers. The Nominal level always has an effort multiplier of 1.00, which does not change the estimated effort. Off-nominal ratings generally do change the estimated effort. For example, a high rating of Required Software Reliability (RELY) will add 10% to the estimated effort, as determined by the COCOMO II.2000 data calibration. A Very High RELY rating will add 26%. It is possible to assign intermediate rating levels and corresponding effort multipliers for your project. For example, the USC COCOMO II software tool supports rating cost drivers between the rating levels in quarter increments, e.g. Low+0.25, Nominal+0.50, High+0.75, etc. Whenever an assessment of a cost driver is halfway between quarter increments always round to the Nominal rating, e.g. if a cost driver rating falls halfway between Low+0.5 and Low+0.75, then select Low+0.75; or if a rating falls halfway between High+0.25 and High+0.5, then select High+0.25. Normally, linear interpolation is used to determine intermediate multiplier values, but nonlinear interpolation is more accurate for the high end of the TIME and STOR cost drivers and the low end of SCED.

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. The unique handling of SCED is discussed in Section 3.2.1.4 and in 4.

3.2.1.1 Product Factors

Product factors account for variation in the effort required to develop software caused by characteristics of the product under development. A product that is complex, has high reliability requirements, or works with a large testing database will require more effort to complete. There are five product factors, and complexity has the strongest influence on estimated effort.

Required Software Reliability (RELY)

This is the measure of the extent to which the software must perform its intended function over a period of time. If the effect of a software failure is only slight inconvenience then RELY is very low. If a failure would risk human life then RELY is very high. Table 17 provides the COCOMOII.2000 rating scheme for RELY.

Table 17. RELY Cost Driver

RELY Descriptors:	slight inconven-ience	low, easily recoverable losses	moderate, easily recoverable losses	high financial loss	risk to human life	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.82	0.92	1.00	1.10	1.26	n/a

This cost driver can be influenced by the requirement to develop software for reusability, see the description for RUSE.

Data Base Size (DATA)

This cost driver attempts to capture the effect large test data requirements have on product development. The rating is determined by calculating D/P, the ratio of bytes in the testing database to SLOC in the program. The reason the size of the database is important to consider is because of the effort required to generate the test data that will be used to exercise the program. In other words, DATA is capturing the effort needed to assemble and maintain the data required to complete test of the program through IOC, see Table 18.

Table 18. DATA Cost Driver

DATA* Descriptors		Testing DB bytes/Pgm SLOC < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P ≥ 1000	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.90	1.00	1.14	1.28	n/a

***** DATA is rated as Low if D/P is less than 10 and it is very high if it is greater than 1000. P is measured in equivalent source lines of code (SLOC), which may involve function point or reuse conversions.

Product Complexity (CPLX)

Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations. Using Table 19, select the area or combination of areas that characterize the product or the component of the product you are rating. The complexity rating is the subjective weighted average of the selected area ratings. Table 20 provides the COCOMO II.2000 effort multipliers for CPLX.

Table 19. Component Complexity Ratings Levels

	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations	User Interface Management Operations
Very Low	Straight-line code with a few non-nested structured programming operators: DOs, CASEs, IF-THEN-ELSEs. Simple module composition via procedure calls or simple scripts.	Evaluation of simple expressions: e.g., A=B+C*(D-E)	Simple read, write statements with simple formats.	Simple arrays in main memory. Simple COTS-DB queries, updates.	Simple input forms, report generators.
Low	Straightforward nesting of structured programming operators. Mostly simple predicates	Evaluation of moderate-level expressions: e.g., D=SQRT(B**2-4.*A*C)	No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level.	Single file subsetting with no data structure changes, no edits, no intermediate files. Moderately complex COTS-DB queries, updates.	Use of simple graphic user interface (GUI) builders.
Nominal	Mostly simple nesting. Some intermodule control. Decision tables. Simple callbacks or message passing, including middleware-supported distributed processing	Use of standard math and statistical routines. Basic matrix/vector operations.	I/O processing includes device selection, status checking and error processing.	Multi-file input and single file output. Simple structural changes, simple edits. Complex COTS-DB queries, updates.	Simple use of widget set.

Table 19. Component Complexity Ratings Levels

	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations	User Interface Management Operations
High	Highly nested structured programming operators with many compound predicates. Queue and stack control. Homogeneous, distributed processing. Single processor soft real-time control.	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, round-off concerns.	Operations at physical I/O level (physical storage address translations; seeks, reads, etc.). Optimized I/O overlap.	Simple triggers activated by data stream contents. Complex data restructuring.	Widget set development and extension. Simple voice I/O, multimedia.
Very High	Reentrant and recursive coding. Fixed-priority interrupt handling. Task synchronization, complex callbacks, heterogeneous distributed processing. Single-processor hard real-time control.	Difficult but structured numerical analysis: near-singular matrix equations, partial differential equations. Simple parallelization.	Routines for interrupt diagnosis, servicing, masking. Communication line handling. Performance-intensive embedded systems.	Distributed database coordination. Complex triggers. Search optimization.	Moderately complex 2D/3D, dynamic graphics, multimedia.
Extra High	Multiple resource scheduling with dynamically changing priorities. Microcode-level control. Distributed hard real-time control.	Difficult and unstructured numerical analysis: highly accurate analysis of noisy, stochastic data. Complex parallelization.	Device timing-dependent coding, micro-programmed operations. Performance-critical embedded systems.	Highly coupled, dynamic relational and object structures. Natural language data management.	Complex multimedia, virtual reality, natural language interface.

Table 20. CPLX Cost Driver

Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.73	0.87	1.00	1.17	1.34	1.74

Developed for Reusability (RUSE)

This cost driver accounts for the additional effort needed to construct components intended for reuse on current or future projects. This effort is consumed with creating more generic design of software, more elaborate documentation, and more extensive testing to ensure components are ready for use in other applications. "Across project" could apply to reuse across the modules in a single financial applications project. "Across program" could apply to reuse across multiple financial applications projects for a single organization. "Across product line" could apply if the reuse is extended across multiple organizations. "Across multiple product lines" could apply to reuse across financial, sales, and marketing product lines, see Table 21.

Development for reusability imposes constraints on the project's RELY and DOCU ratings. The RELY rating should be at most one level below the RUSE rating. The DOCU rating should be at least Nominal for Nominal and High RUSE ratings, and at least High for Very High and Extra High RUSE ratings.

Table 21. RUSE Cost Driver

RUSE Descriptors:		none	across project	across program	across product line	across multiple product lines
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.95	1.00	1.07	1.15	1.24

Documentation Match to Life-Cycle Needs (DOCU)

Several software cost models have a cost driver for the level of required documentation. In COCOMO II, the rating scale for the DOCU cost driver is evaluated in terms of the suitability of the project's documentation to its life-cycle needs. The rating scale goes from Very Low (many life-cycle needs uncovered) to Very High (very excessive for life-cycle needs), see Table 22.

Attempting to save costs via Very Low or Low documentation levels will generally incur extra costs during the maintenance portion of the life-cycle. Poor or missing documentation will increase the Software Understanding (SU) increment discussed in Section 2.4.2.

Table 22. DOCU Cost Driver

DOCU Descriptors:	Many life-cycle needs uncovered	Some life-cycle needs uncovered.	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very excessive for life-cycle needs	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.81	0.91	1.00	1.11	1.23	n/a

This cost driver can be influenced by the developed for reusability cost factor, see the description for RUSE.

3.2.1.2 Platform Factors

The platform refers to the target-machine complex of hardware and infrastructure software (previously called the virtual machine). The factors have been revised to reflect this as described in this section. Some additional platform factors were considered, such as distribution, parallelism, embeddedness, and real-time operations. These considerations have been accommodated by the expansion of the Component Complexity rating levels in Table 19.

Execution Time Constraint (TIME)

This is a measure of the execution time constraint imposed upon a software system. The rating is expressed in terms of the percentage of available execution time expected to be used by the system or subsystem consuming the execution time resource. The rating ranges from nominal, less than 50% of the execution time resource used, to extra high, 95% of the execution time resource is consumed, see Table 23.

Table 23. TIME Cost Driver

TIME Descriptors:			≤ 50% use of available execution time	70% use of available execution time	85% use of available execution time	95% use of available execution time
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1.00	1.11	1.29	1.63

Main Storage Constraint (STOR)

This rating represents the degree of main storage constraint imposed on a software system or subsystem. Given the remarkable increase in available processor execution time and main storage, one can question whether these constraint variables are still relevant. However, many applications continue to expand to consume whatever resources are available---particularly with large and growing COTS products---making these cost drivers still relevant. The rating ranges from nominal (less than 50%), to extra high (95%) see Table 24.

Table 24. STOR Cost Driver

STOR Descriptors:			≤ 50% use of available storage	70% use of available storage	85% use of available storage	95% use of available storage
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	n/a	1.00	1.05	1.17	1.46

Platform Volatility (PVOL)

"Platform" is used here to mean the complex of hardware and software (OS, DBMS, etc.) the software product calls on to perform its tasks. If the software to be developed is an operating system then the platform is the computer hardware. If a database management system is to be developed then the platform is the hardware and the operating system. If a network text browser is to be developed then the platform is the network, computer hardware, the operating system,

and the distributed information repositories. The platform includes any compilers or assemblers supporting the development of the software system. This rating ranges from low, where there is a major change every 12 months, to very high, where there is a major change every two weeks, see Table 25.

Table 25. PVOL Cost Driver

PVOL Descriptors:		Major change every 12 mo.; Minor change every 1 mo.	Major: 6 mo.; Minor: 2 wk.	Major: 2 mo.;Minor: 1 wk.	Major: 2 wk.;Minor: 2 days	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	n/a	0.87	1.00	1.15	1.30	n/a

3.2.1.3 Personnel Factors

After product size, people factors have the strongest influence in determining the amount of effort required to develop a software product. The Personnel Factors are for rating the development team's capability and experience – not the individual. These ratings are most likely to change during the course of a project reflecting the gaining of experience or the rotation of people onto and off the project.

Analyst Capability (ACAP)

Analysts are personnel who work on requirements, high-level design and detailed design. The major attributes that should be considered in this rating are analysis and design ability, efficiency and thoroughness, and the ability to communicate and cooperate. The rating should not consider the level of experience of the analyst; that is rated with APEX, LTEX, and PLEX. Analyst teams that fall in the fifteenth percentile are rated very low and those that fall in the ninetieth percentile are rated as very high, see Table 26.

Table 26. ACAP Cost Driver

ACAP Descriptors:	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.42	1.19	1.00	0.85	0.71	n/a

Programmer Capability (PCAP)

Current trends continue to emphasize the importance of highly capable analysts. However the increasing role of complex COTS packages, and the significant productivity leverage associated with programmers' ability to deal with these COTS packages, indicates a trend toward higher importance of programmer capability as well.

Evaluation should be based on the capability of the programmers as a team rather than as individuals. Major factors which should be considered in the rating are ability, efficiency and thoroughness, and the ability to communicate and cooperate. The experience of the programmer

should not be considered here; it is rated with APEX, LTEX, and PLEX. A very low rated programmer team is in the fifteenth percentile and a very high rated programmer team is in the ninetieth percentile, see Table 27.

Table 27. PCAP Cost Driver

PCAP Descriptors	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.34	1.15	1.00	0.88	0.76	n/a

Personnel Continuity (PCON)

The rating scale for PCON is in terms of the project's annual personnel turnover: from 3%, very high continuity, to 48%, very low continuity, see Table 28.

Table 28. PCON Cost Driver

PCON Descriptors:	48% / year	24% / year	12% / year	6% / year	3% / year	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.29	1.12	1.00	0.90	0.81	

Applications Experience (APEX)

The rating for this cost driver (formerly labeled AEXP) is dependent on the level of applications experience of the project team developing the software system or subsystem. The ratings are defined in terms of the project team's equivalent level of experience with this type of application. A very low rating is for application experience of less than 2 months. A very high rating is for experience of 6 years or more, see Table 29.

Table 29. APEX Cost Driver

APEX Descriptors:	≤ 2 months	6 months	1 year	3 years	6 years	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.22	1.10	1.00	0.88	0.81	n/a

Platform Experience (PLEX)

The Post-Architecture model broadens the productivity influence of platform experience, PLEX (formerly labeled PEXP), by recognizing the importance of understanding the use of more powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities, see Table 30.

Table 30. PLEX Cost Driver

PLEX Descriptors:	≤ 2 months	6 months	1 year	3 years	6 year	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.19	1.09	1.00	0.91	0.85	n/a

Language and Tool Experience (LTEX)

This is a measure of the level of programming language and software tool experience of the project team developing the software system or subsystem. Software development includes the use of tools that perform requirements and design representation and analysis, configuration management, document extraction, library management, program style and formatting, consistency checking, planning and control, etc. In addition to experience in the project's programming language, experience on the project's supporting tool set also affects development effort. A low rating is given for experience of less than 2 months. A very high rating is given for experience of 6 or more years, see Table 31.

Table 31. LTEX Cost Driver

LTEX Descriptors:	≤ 2 months	6 months	1 year	3 years	6 year	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.20	1.09	1.00	0.91	0.84	

3.2.1.4 Project Factors

Project factors account for influences on the estimated effort such as use of modern software tools, location of the development team, and compression of the project schedule.

Use of Software Tools (TOOL)

Software tools have improved significantly since the 1970s' projects used to calibrate the 1981 version of COCOMO. The tool rating ranges from simple edit and code, very low, to integrated life-cycle management tools, very high. A Nominal TOOL rating in COCOMO 81 is equivalent to a Very Low TOOL rating in COCOMO II. An emerging extension of COCOMO II is in the process of elaborating the TOOL rating scale and breaking out the effects of TOOL capability, maturity, and integration, see Table 32.

Table 32. TOOL Cost Driver

TOOL Descriptors	edit, code, debug	simple, frontend, backend CASE, little integration	basic life-cycle tools, moderately integrated	strong, mature life-cycle tools, moderately integrated	strong, mature, proactive life-cycle tools, well integrated with processes, methods, reuse	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.17	1.09	1.00	0.90	0.78	n/a

Multisite Development (SITE)

Given the increasing frequency of multisite developments, and indications that multisite development effects are significant, the SITE cost driver has been added in COCOMO II. Determining its cost driver rating involves the assessment and judgement-based averaging of two factors: site collocation (from fully collocated to international distribution) and communication support (from surface mail and some phone access to full interactive multimedia).

For example, if a team is fully collocated, it doesn't need interactive multimedia to achieve an Extra High rating. Narrowband e-mail would usually be sufficient, see Table 33.

Table 33. SITE Cost Driver

SITE: Collocation Descriptors:	Inter-national	Multi-city and Multi-company	Multi-city or Multi-company	Same city or metro. area	Same building or complex	Fully collocated
SITE: Communications Descriptors:	Some phone, mail	Individual phone, FAX	Narrow band email	Wideband electronic communicat ion.	Wideband elect. comm., occasional video conf.	Interactive multimedia
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.22	1.09	1.00	0.93	0.86	0.80

Required Development Schedule (SCED)

This rating measures the schedule constraint imposed on the project team developing the software. The ratings are defined in terms of the percentage of schedule stretch-out or acceleration with respect to a nominal schedule for a project requiring a given amount of effort. Accelerated schedules tend to produce more effort in the earlier phases to eliminate risks and refine the architecture, more effort in the later phases to accomplish more testing and documentation in parallel. In Table 34, schedule compression of 75% is rated very low. A schedule stretch-out of 160% is rated very high. Stretch-outs do not add or decrease effort. Their savings because of smaller team size are generally balanced by the need to carry project administrative functions over a longer period of time. The nature of this balance is undergoing

further research in concert with our emerging CORADMO extension to address rapid application development (goto http://sunset.usc.edu/COCOMOII/suite.html for more information).

SCED is the only cost driver that is used to describe the effect of schedule compression / expansion *for the whole project*. The scale factors are also used to describe the whole project. All of the other cost drivers are used to describe each module in a multiple module project. Using the COCOMO II Post-Architecture model for multiple module estimation is explained in Section 3.3.

Table 34. SCED Cost Driver

SCED Descriptors	75% of nominal	85% of nominal	100% of nominal	130% of nominal	160% of nominal	
Rating Level	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multiplier	1.43	1.14	1.00	1.00	1.00	n/a

SCED is also handled differently in the COCOMO II estimation of time to develop, TDEV. This special use of SCED is explained in Section 4.

3.2.2 Early Design Model Drivers

This model is used in the early stages of a software project when very little may be known about the size of the product to be developed, the nature of the target platform, the nature of the personnel to be involved in the project, or the detailed specifics of the process to be used. This model could be employed in either Application Generator, System Integration, or Infrastructure development sectors. For discussion of these marketplace sectors see [Boehm et al. 2000].

The Early Design model uses KSLOC or unadjusted function points (UFP) for size. UFPs are converted to the equivalent SLOC and then to KSLOC as discussed in Section 2.3. The application of exponential scale factors is the same for Early Design and the Post-Architecture models and was described in Section 3.1. In the Early Design model a reduced set of multiplicative cost drivers is used as shown in Table 35. The Early Design cost drivers are obtained by combining the Post-Architecture model cost drivers. Whenever an assessment of a cost driver is halfway between the rating levels always round to the Nominal rating, e.g. if a cost driver rating is halfway between Very Low and Low, then select Low. The effort equation is the same as given in Equation 11 except that the number of effort multipliers is reduced to 7 (n = 7).

Table 35. Early Design and Post-Architecture Effort Multipliers

Early Design Cost Driver	Counterpart Combined Post-Architecture Cost Drivers
PERS	ACAP, PCAP, PCON
RCPX	RELY, DATA, CPLX, DOCU
RUSE	RUSE
PDIF	TIME, STOR, PVOL
PREX	APEX, PLEX, LTEX
FCIL	TOOL, SITE
SCED	SCED

Overall Approach

The following approach is used for mapping the full set of Post-Architecture cost drivers and rating scales onto their Early Design model counterparts. It involves the use and combination of numerical equivalents of the rating levels. Specifically, a Very Low Post-Architecture cost driver rating corresponds to a numerical rating of 1, Low is 2, Nominal is 3, High is 4, Very High is 5, and Extra High is 6. For the combined Early Design cost drivers, the numerical values of the contributing Post-Architecture cost drivers are summed, and the resulting totals are allocated to an expanded Early Design model rating scale going from Extra Low to Extra High. The Early Design model rating scales always have a Nominal total equal to the sum of the Nominal ratings of its contributing Post-Architecture elements. An example is given below for the PERS cost driver.

Personnel Capability (PERS) and Mapping Example

An example will illustrate this approach. The Early Design PERS cost driver combines the Post-Architecture cost drivers Analyst capability (ACAP), Programmer capability (PCAP), and Personnel continuity (PCON), see Table 36. Each of these has a rating scale from Very Low (=1) to Very High (=5). Adding up their numerical ratings produces values ranging from 3 to 15. These are laid out on a scale, and the Early Design PERS rating levels assigned to them, as shown below. The associated effort multipliers are derived from the ACAP, PCAP, and PCON effort multipliers by averaging the products of each combination of effort multipliers associated with the given Early Design rating level.

The effort multipliers for PERS, like the other Early Design model cost drivers, are derived from those of the Post-Architecture model by averaging the products of the constituent Post-Architecture multipliers (in this case ACAP, PCAP, PCON) for each combination of cost driver ratings corresponding with the Early Design rating level. For PERS = Extra High, this would involve four combinations: ACAP, PCAP, and PCON all Very High, or only one High and the other two Very High.

Table 36. PERS Cost Driver

PERS Descriptors:							
• Sum of ACAP, PCAP, PCON Ratings	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
• Combined ACAP and PCAP Percentile	20%	35%	45%	55%	65%	75%	85%

Table 36. PERS Cost Driver

PERS Descriptors:							
• Annual Personnel Turnover	45%	30%	20%	12%	9%	6%	4%
Rating Levels	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	2.12	1.62	1.26	1.00	0.83	0.63	0.50

The Nominal PERS rating of 9 corresponds to the sum (3 + 3 + 3) of the Nominal ratings for ACAP, PCAP, and PCON, and its corresponding effort multiplier is 1.0. Note, however that the Nominal PERS rating of 9 can result from a number of other combinations, e.g. 1 + 3 + 5 = 9 for ACAP = Very Low, PCAP = Nominal, and PCON = Very High.

The rating scales and effort multipliers for PCAP and the other Early Design cost drivers maintain consistent relationships with their Post-Architecture counterparts. For example, the PERS Extra Low rating levels (20% combined ACAP and PCAP percentile; 45% personnel turnover) represent averages of the ACAP, PCAP, and PCON rating levels adding up to 3 or 4.

Maintaining these consistency relationships between the Early Design and Post-Architecture rating levels ensures consistency of Early Design and Post-Architecture cost estimates. It also enables the rating scales for the individual Post-Architecture cost drivers, Table 35, to be used as detailed backups for the top-level Early Design rating scales given above.

Product Reliability and Complexity (RCPX)

This Early Design cost driver combines the four Post-Architecture cost drivers Required software reliability (RELY), Database size (DATA), Product complexity (CPLX), and Documentation match to life-cycle needs (DOCU). Unlike the PERS components, the RCPX components have rating scales with differing width. RELY and DOCU range from Very Low to Very High; DATA ranges from Low to Very High, and CPLX ranges from Very Low to Extra High. The numerical sum of their ratings thus ranges from 5 (VL, L, VL, VL) to 21 (VH, VH, EH, VH).

Table 36 assigns RCPX ratings across this range, and associates appropriate rating scales to each of the RCPX ratings from Extra Low to Extra High. As with PERS, the Post-Architecture RELY, DATA CPLX, and DOCU rating scales discussed in Section 3.2.1.1 provide detailed backup for interpreting the Early Design RCPX rating levels.

Table 37. RCPX Cost Driver

RCPX Descriptors:							
• Sum of RELY, DATA, CPLX, DOCU Ratings	5, 6	7, 8	9 - 11	12	13 - 15	16 - 18	19 - 21
• Emphasis on reliability, documentation	Very Little	Little	Some	Basic	Strong	Very Strong	Extreme
• Product complexity	Very simple	Simple	Some	Moderate	Complex	Very complex	Extremely complex
• Database size	Small	Small	Small	Moderate	Large	Very Large	Very Large

Table 37. RCPX Cost Driver

RCPX Descriptors:							
Rating Levels	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.49	0.60	0.83	1.00	1.33	1.91	2.72

Developed for Reusability (RUSE)

This Early Design model cost driver is the same as its Post-Architecture counterpart, which is covered in Section 3.2.1

Platform Difficulty (PDIF)

This Early Design cost driver combines the three Post-Architecture cost drivers Execution time constraint (TIME), Main storage constraint (STOR), and Platform volatility (PVOL). TIME and STOR range from Nominal to Extra High; PVOL ranges from Low to Very High. The numerical sum of their ratings thus ranges from 8 (N, N, L) to 17 (EH, EH, VH).

Table 38 assigns PDIF ratings across this range, and associates the appropriate rating scales to each of the PDIF rating levels. The Post-Architecture rating scales in Tables 23, 24, 25 provide additional backup definition for the PDIF ratings levels.

Table 38. PDIF Cost Driver

PDIF Descriptors:					
• Sum of TIME, STOR, and PVOL ratings	8	9	10 - 12	13 - 15	16, 17
• Time and storage constraint	≤ 50%	≤ 50%	65%	80%	90%
• Platform volatility	Very stable	Stable	Somewhat volatile	Volatile	Highly volatile
Rating Levels	Low	Nominal	High	Very High	Extra High
Effort Multipliers	0.87	1.00	1.29	1.81	2.61

Personnel Experience (PREX)

This Early Design cost driver combines the three Post-Architecture cost drivers Application experience (APEX), Language and tool experience (LTEX), and Platform experience (PLEX). Each of these range from Very Low to Very High; as with PERS, the numerical sum of their ratings ranges from 3 to 15.

Table 39 assigns PREX ratings across this range, and associates appropriate effort multipliers and rating scales to each of the rating levels.

Table 39. PREX Cost Driver

PREX Descriptors:							
• Sum of APEX, PLEX, and LTEX ratings	3, 4	5, 6	7, 8	9	10, 11	12, 13	14, 15
• Applications, Platform, Language and Tool Experience	≤ 3 mo.	5 months	9 months	1 year	2 years	4 years	6 years

Table 39. PREX Cost Driver

PREX Descriptors:							
Rating Levels	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.59	1.33	1.22	1.00	0.87	0.74	0.62

Facilities (FCIL)

This Early Design cost driver combines two Post-Architecture cost drivers: Use of software tools (TOOL) and Multisite development (SITE). TOOL ranges from Very Low to Very High; SITE ranges from Very Low to Extra High. Thus, the numerical sum of their ratings ranges from 2 (VL, VL) to 11 (VH, EH).

Table 40 assigns FCIL ratings across this range, and associates appropriate rating scales to each of the FCIL rating levels. The individual Post-Architecture TOOL and SITE rating scales in Section 3.2.1 again provide additional backup definition for the FCIL rating levels.

Table 40. FCIL Cost Driver

FCIL Descriptors:							
• Sum of TOOL and SITE ratings	2	3	4, 5	6	7, 8	9, 10	11
• TOOL support	Minimal	Some	Simple CASE tool collection	Basic life-cycle tools	Good; moderately integrated	Strong; moderately integrated	Strong; well integrated
• Multisite conditions	Weak support of complex multisite development	Some support of complex M/S devel.	Some support of moderately complex M/S devel.	Basic support of moderately complex M/S devel.	Strong support of moderately complex M/S devel.	Strong support of simple M/S devel.	Very strong support of collocated or simple M/S devel.
Rating Levels	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.43	1.30	1.10	1.0	0.87	0.73	0.62

Required Development Schedule (SCED)

This Early Design model cost driver is the same as its Post-Architecture counterpart, which is covered in Section 3.2.1.

3.3 Multiple Module Effort Estimation

Usually software systems are comprised of multiple subsystems or components. It is possible to use COCOMO II to estimate effort and schedule for multiple components. The technique described here is for one level of sub-components. For multiple levels of sub-components see [Boehm 1981].

The COCOMO II method for doing this does not use the sum of the estimates for each component as this would ignore effort due to integration of the components. The COCOMO II multiple module method for n number of modules has the following steps:

1. Sum the sizes for all of the components, $Size_i$, to yield an aggregate size.

$$Size_{Aggregate} = \sum_{i=1}^{n} Size_i$$

2. Apply the project-level drivers, the scale factors and the SCED Cost Driver, to the aggregated size to derive the overall basic effort for the total project, PM_{Basic}. The scale factors are discussed in Section 2.3.1 and SCED is discussed in Section 2.3.2.1.

$$PM_{Basic} = A \times (Size_{Aggregate})^{E} \times SCED$$

3. Determine each component's basic effort, $PM_{Basic(i)}$, by apportioning the overall basic effort to each component based on its contribution to the aggregate size.

$$PM_{Basic(i)} = PM_{Basic} \times \left(\frac{Size_i}{Size_{Aggregate}} \right)$$

4. Apply the component-level Cost Drivers (excluding SCED) to each component's basic effort.

$$PM_i = PM_{Basic(i)} \times \sum_{j=1}^{16} EM_j$$

5. Sum each component's effort to derive the aggregate effort, $PM_{Aggregate}$, for the total project.

$$PM_{Aggregate} = \sum_{i=1}^{n} PM_i$$

6. The schedule is estimated by repeating steps 2 through 5 without the SCED Cost Driver used in step 2. Using this modified aggregate effort, $PM'_{Aggregate}$, the schedule is derive using Equation 14 in Section 4.

4. Schedule Estimation

The initial version of COCOMO II provides a simple schedule estimation capability similar to those in COCOMO 81 and Ada COCOMO. The initial baseline schedule equation for the COCOMO II Early Design and Post-Architecture stages is:

$$\text{TDEV} = [\text{C} \times (\text{PM}_{\text{NS}})^{(\text{D}+0.2\times(\text{E}-\text{B}))}] \times \frac{\text{SCED\%}}{100} \qquad \textbf{Eq. 14}$$

$$\text{where C} = 3.67, \text{D} = 0.28, \text{B} = 0.91$$

In Equation 14, C is a TDEV coefficient that can be calibrated, PM_{NS} is the estimated PM *excluding* the SCED effort multiplier as defined in Equation 1, D is a TDEV scaling base-exponent that can also be calibrated. E is the effort scaling exponent derived as the sum of project scale factors and B as the calibrated scale factor base-exponent (discussed in Sections 3.1). SCED% is the compression / expansion percentage in the SCED effort multiplier rating scale discussed in Section 3.2.1.

Time to Develop, TDEV, is the calendar time in months between the estimation end points of LCO and IOC for MBASE/RUP or SRR and SAR for Waterfall lifecycle models (see Section 6.2). For the waterfall model, this goes from the determination of a product's requirements baseline to the completion of an acceptance activity certifying that the product satisfies its requirements. For the MBASE/RUP model discussed in Section 6, it covers the time span between LCO and IOC milestones.

As COCOMO II evolves, it will have a more extensive schedule estimation model, reflecting the different classes of process models a project can use. The effects of reusable and COTS software; the effects of applications composition capabilities; and the effects of alternative strategies such as Rapid Application Development are discussed in [Boehm et al. 2000; also at http://sunset.usc.edu/COOCMOII/suite.html].

5. Software Maintenance

Software maintenance is defined as the process of modifying existing software while not changing its primary functions [Boehm 1981]. The assumption made by the COCOMO II model is that software maintenance cost generally has the same cost driver attributes as software development costs. Maintenance includes redesign and recoding of small portions of the original product, redesign and development of interfaces, and minor modification of the product structure. Maintenance can be classified as either updates or repairs. Product repairs can be further segregated into corrective (failures in processing, performance, or implementation), adaptive (changes in the processing or data environment), or perfective maintenance (enhancing performance or maintainability). Maintenance sizing is covered in Section 2.7.

There are special considerations for using COCOMO II in software maintenance. Some of these are adapted from [Boehm 1981].

- The SCED cost driver (Required Development Schedule) is not used in the estimation of effort for maintenance. This is because the maintenance cycle is usually of a fixed duration.
- The RUSE cost driver (Required Reusablity) is not used in the estimation of effort for maintenance. This is because the extra effort required to maintain a component's reusability is roughly balanced by the reduced maintenance effort due to the component's careful design, documentation, and testing.
- The RELY cost driver (Required Software Reliabilty) has a different set of effort multipliers for maintenance. For maintenance the RELY cost driver depends on the required reliability under which the product was developed. If the product was developed with low reliability it will require more effort to fix latent faults. If the product was developed with very high reliability, the effort required to maintain that level of reliability will be above nominal. Table 41 below shows the effort multipliers for RELY.

Table 41. RELY Maintenance Cost Driver

RELY Descriptors:	slight inconvenience	low, easily recoverable losses	moderate, easily recoverable losses	high financial loss	risk to human life	
Rating Levels	Very Low	Low	Nominal	High	Very High	Extra High
Effort Multipliers	1.23	1.10	1.00	0.99	1.07	n/a

- The scaling exponent, E, is applied to the number of changed KSLOC (added and modified, not deleted) rather than the total legacy system KSLOC. As discussed in Section 2.7, the effective maintenance size $(Size)_m$ is adjusted by a Maintenance Adjustment Factor (MAF) to account for legacy system effects.

The maintenance effort estimation formula is the same as the COCOMO II Post-Architecture development model (with the exclusion of SCED and RUSE):

$$PM_M = A \times (Size_M)^E \times \prod_{i=1}^{15} EM_i \qquad \textbf{Eq. 15}$$

The COCOMO II approach differs from the COCOMO 81 maintenance effort estimation by letting you use any desired maintenance activity duration, TM. The average maintenance staffing level, FSPM, can then be obtained via the relationship:

$$FSPM = PM_M / TM \qquad \textbf{Eq. 16}$$

6. COCOMO II: Assumptions and phase/activity distributions

6.1 Introduction

Section defines the particular COCOMO II assumptions about what life-cycle phases and labor categories are covered by its effort and schedule estimates. These and other definitions given in Section 6 were used in collecting all the data to which COCOMO II has been calibrated. If you use other definitions and assumptions, you need to either adjust the COCOMO II estimates or recalibrate its coefficients.

COCOMO II has been developed to be usable by projects employing either waterfall or spiral processes. For these to be reasonably compatible, the waterfall implementation needs to be strongly risk-driven, in order to avoid incurring large amounts of rework not included in spiral-model-based estimates. Fortunately, this was the case for the normative waterfall implementation provided in Chapter 4 of [Boehm 1981] as the underlying process model for COCOMO 81.

The implementation of the spiral model used by COCOMO II also needs an added feature: a set of well-defined common milestones which can serve as the end points between which COCOMO II estimates and actuals are assessed. In 1995, we devoted parts of two COCOMO II Affiliates' workshops to determining such milestones. The result was the set of Anchor Point milestones: Life Cycle Objectives (LCO), Life Cycle Architecture (LCA), and Initial Operational Capability (IOC) [Boehm 1996]. Those milestones were a good fit to key life-cycle project commitment points being used within both our commercial and government contractor Affiliate communities. The LCO and LCA milestones involve concurrent rather than sequential development and elaboration of a system's operational concept, requirements, architecture, prototypes, life-cycle plan, and feasibility rationale. The milestones correspond well with real-life commitment milestones: LCO is roughly equivalent to getting engaged to your system definition; LCA to getting married; and IOC to having your first child.

These anchor points and the stakeholder win-win extension of the spiral model became the key milestones in our Model-Based (System) Architecting and Software Engineering (MBASE) life-cycle process model [Boehm-Port 1999; Boehm et al. 1999]. We have also collaborated with one of our Affiliates, Rational, Inc., to ensure the compatibility of MBASE and the Rational Unified Process (RUP). Thus, we have adopted Rational's approach to the four main spiral-oriented phases: Inception, Elaboration, Construction, and Transition. Rational has adopted our definitions of the LCO, LCA, and IOC anchor point milestones defining the entry and exit criteria between the phases [Royce 1998; Kruchten 1999; Jacobson et al. 1999].

Section 6.2 proceeds to define the content of the milestones used as end points for the waterfall and MBASE/RUP spiral models to which COCOMO II project estimates are related. Section 6.3 compares the phase distributions of effort and schedule used by COCOMO II for the waterfall and initial MBASE/RUP spiral process models. Section 6.4 defines the activity categories for the Waterfall and MBASE/RUP spiral models, and their content. Section 6.5 presents the corresponding effort distributions by activity for each Waterfall and MBASE/RUP phase. Section 6.6 covers other COCOMO II assumptions, such as the labor categories considered as "project effort," and the number of person-hours in a person-month.

6.2 Waterfall and MBASE/RUP Phase Definitions

6.2.1 Waterfall Model Phases and Milestones

Table 42 defines the milestones used as end points for COCOMO II Waterfall phase effort and schedule estimates. The milestone definitions are the same as those in [Boehm 1981; Table 4-1].

A basic risk-orientation is provided with the inclusion of "Identification and resolution of all high-risk development issues" as a Product Design milestone element. However, the other early milestones should have a more risk-driven interpretation. For example, having "…specifications validated for…feasibility" by the end of the Plans and Requirements phase of a user-interactive system development would imply doing an appropriate amount of user-interface prototyping.

Table 42. COCOMO II Waterfall Milestones

1. Begin Plans and Requirements Phase. (Completion of Life Cycle Concept Review - LCR) ▪ Approved, validated system architecture, including basic hardware-software allocations. ▪ Approved, validated concept of operation, including basic human-machine allocations. ▪ Top-level life-cycle plan, including milestones, resources, responsibilities, schedules, and major activities.
2. End Plans and Requirements Phase. Begin Product Design Phase. (Completion of Software Requirements Review - SRR) ▪ Detailed development plan – detailed development milestone criteria, resource budgets, organization, responsibilities, schedules, activities, techniques, and products. ▪ Detailed usage plan – counterparts of the development plan items for training, conversion, installation, operations, and support. ▪ Detailed product control plan – configuration management plan, quality assurance plan, overall V&V plan (excluding detailed test plans). ▪ Approved, validated software requirements specifications – functional, performance, and interface specifications validated for completeness, consistency, testability, and feasibility. ▪ Approved (formal or informal) development contract – based on the above items.
3. End Product Design Phase. Begin Detailed Design Phase. (Completion of Product Design Review - PDR) ▪ Verified software product design specification. ▪ Program component hierarchy, control and data interfaces through unit level*. ▪ Physical and logical data structure through field level. ▪ Data processing resource budgets (timing, storage, accuracy). ▪ Verified for completeness, consistency, feasibility, and traceability to requirements. ▪ Identification and resolution of all high-risk development issues. ▪ Preliminary integration and test plan, acceptance test plan, and user's manual.
4. End Detailed Design Phase. Begin Code and Unit Test Phase. (Completion of design walkthrough or Critical Design Review for unit - CDR) ▪ Verified detailed design specification for each unit. ▪ For each routine ($\leq$ 100 source instructions) within the unit, specifies name, purpose, assumptions, sizing, calling sequence, error exits, inputs, outputs, algorithms, and processing flow. ▪ Data base description through parameter/character/bit level. ▪ Verified for completeness, consistency, and traceability to requirements and system design specifications and budgets. ▪ Approved acceptance test plan. ▪ Complete draft of integration and test plan and user's manual.

Table 42. COCOMO II Waterfall Milestones

5. End Code and Unit Test Phase. Begin Integration and Test Phase. (Satisfaction of Unit Test criteria for unit - UTC) ▪ Verification of all unit computations, using not only nominal values but also singular and extreme values. ▪ Verification of all unit input and output options, including error messages. ▪ Exercise of all executable statements and all branch options. ▪ Verification of programming standards compliance. ▪ Completion of unit-level, as-built documentation.
6. End Integration and Test Phase. Begin Implementation Phase. (Completion of Software Acceptance Review - SAR) ▪ Satisfaction of software acceptance test. ▪ Verification of satisfaction of software requirements. ▪ Demonstration of acceptable off-nominal performance as specified. ▪ Acceptance of all deliverable software products: reports, manuals, as-built specifications, data bases.
7. End Implementation Phase. Begin Operations and Maintenance Phase. (Completion of System Acceptance Review) ▪ Satisfaction of system acceptance test. ▪ Verification of satisfaction of system requirements. ▪ Verification of operational readiness of software, hardware, facilities, and personnel. ▪ Acceptance of all deliverable system products: hardware, software, documentation, training, and facilities. ▪ Completion of all specified conversion and installation activities.
8. End Operations and Maintenance Phase (via Phaseout). ▪ Completion of all items in phaseout plan: conversion, documentation, archiving, transition to new system(s).

* A software unit performs a single well-defined function, can be developed by one person, and is typically 100 to 300 source instructions in size.

6.2.2 MBASE and Rational Unified Process (RUP) Phases and Milestones

Table 43 defines the milestones used as end points for COCOMO II MBASE/RUP phase effort and schedule estimates (the content of the Life Cycle Objectives (LCO) and Life Cycle Architecture (LCA) milestones are elaborated in Table 44). The definitions of the Inception Readiness Review (IRR) and Product Release Review (PRR) have been added in Table 43. They ensure that the Inception and Transition phases have milestones at each end between which to measure effort and schedule. The PRR is defined consistently with its Rational counterpart in [Royce 1998; Kruchten 1999]. The IRR was previously undefined; its content focuses on the preconditions for a successful Inception phase.

Table 43. MBASE and Rational Unified Software Development Process Milestones

1. Inception Readiness Review (IRR) ▪ Candidate system objectives, scope, boundary ▪ Key stakeholders identified ▪ Committed to support Inception phase ▪ Resources committed to achieve successful LCO package

Table 43. MBASE and Rational Unified Software Development Process Milestones

2. Life Cycle Objectives Review (LCO) ▪ Life Cycle Objectives (LCO) Package (see Table 44) ▪ Key elements of Operational Concept, Prototype, Requirements, Architecture, Life Cycle Plan, Feasibility Rationale ▪ Feasibility assured for at least one architecture, using the criteria: ▪ Acceptable business case ▪ A system developed from the architecture would support the operational concept, be compatible with the prototype, satisfy the requirements, and be buildable within the budgets and schedules in the life-cycle plan. ▪ Feasibility validated by an Architecture Review Board (ARB) ▪ ARB includes project-leader peers, architects, specialty experts, key stakeholders [Marenzano 1995]. ▪ Key stakeholders concur on essentials, commit to support Elaboration phase ▪ Resources committed to achieve successful LCA package
3. Life Cycle Architecture Review (LCA) ▪ Life Cycle Architecture (LCA) Package (see Table 44) ▪ Feasibility assured for selected architecture, using the LCO feasibility criteria ▪ Feasibility validated by ARB ▪ Stakeholders concur on their success-critical items, commit to support Construction, Transition, and Maintenance phases. ▪ All major risks resolved or covered by risk management plan ▪ Resources committed to achieve Initial Operational Capability (IOC), life-cycle support
4. Initial Operational Capability (IOC) ▪ Software preparation, including both operational and support software with appropriate commentary and documentation; initial data preparation or conversion; the necessary licenses and rights for COTS and reused software, and appropriate operational readiness testing. ▪ Site preparation, including initial facilities, equipment, supplies, and COTS vendor support arrangements. ▪ Initial user, operator and maintainer preparation, including selection, teambuilding, training and other qualification for familiarization usage, operations, or maintenance. ▪ Successful Transition Readiness Review ▪ Plans, preparations for full conversion, installation, training, and operational cutover ▪ Stakeholders confirm commitment to support Transition and Maintenance phases
5. Product Release Review (PRR) ▪ Assurance of successful cutover from previous system for key operational sites ▪ Personnel fully qualified to operate and maintain new system ▪ Stakeholder concurrence that the deployed system operates consistently with negotiated and evolving stakeholder agreements ▪ Stakeholders confirm commitment to support Maintenance phase

Table 44. Detailed LCO and LCA Milestone Content

Milestone Element	Life Cycle Objectives (LCO)	Life Cycle Architecture (LCA)
Definition of Operational Concept	Top-level system objectives and scope System boundary Environment parameters and assumptions Current system shortfalls Operational concept: key nominal scenarios, stakeholder roles and responsibilities	Elaboration of system objectives and scope by increment Elaboration of operational concept by increment Nominal and key off-nominal scenarios
System Prototype(s)	Exercise key usage scenarios Resolve critical risks	Exercise range of usage scenarios Resolve major outstanding risks
Definition of System and Software Requirements	Top-level capabilities, interfaces, quality attribute levels, including: Evolution requirements Priorities Stakeholders' concurrence on essentials	Elaboration of functions, interfaces, quality attributes by increment Identification of TBDs (to-be-determined items), evolution requirements Stakeholders' concurrence on their priority concerns
Definition of System and Software Architecture	Top-level definition of at least one feasible architecture Physical and logical elements and relationships Choices of COTS and reusable software elements Identification of infeasible architecture options	Choice of architecture and elaboration by increment Physical and logical components, connectors, configurations, constraints COTS, reuse choices Domain-architecture and architectural style choices Architecture evolution parameters
Definition of Life cycle Plan	Identification of life-cycle stakeholders Users, customers, developers, maintainers, interfacers, general public, others Identification of life-cycle process model Top-level stages, increments Top-level WWWWWHH* by stage	Elaboration of WWWWWHH for Initial Operational Capability (IOC) Partial elaboration, identification of key TBDs for later increments
Feasibility Rationale	Assurance of consistency among elements above via analysis, measurement, prototyping, simulation, etc. Business case analysis for requirements, feasible architectures	Assurance of consistency among elements above Rationale for major options rejected All major risks resolved or covered by risk management plan within the life-cycle plan

* WWWWWHH: Why, What, When, Who, Where, How, How Much

Figure 4 shows the relationship of the Waterfall and MBASE/RUP phases and the most likely COCOMO II model to be used in estimating effort and schedule. The milestones have some variation due to the differences in distribution of effort and schedule between the two models.

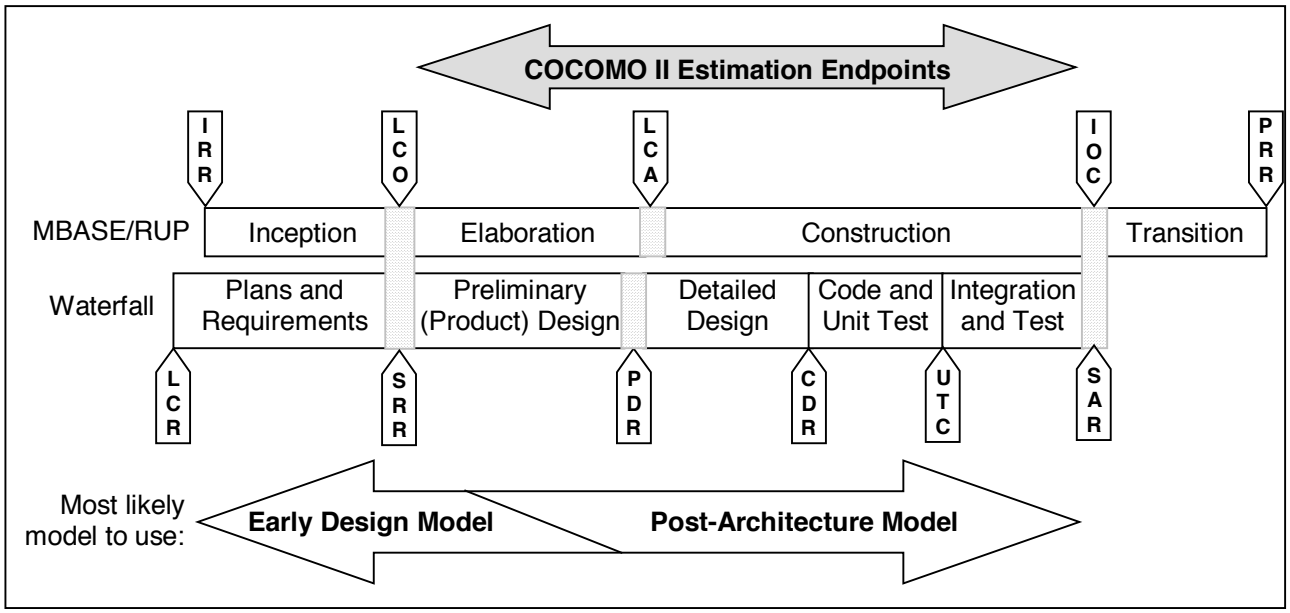


Figure 4. Life Cycle Phases

6.3 Phase Distribution of Effort and Schedule

Provisional phase distributions of effort and schedule are provided below for both the Waterfall and MBASE/RUP process models. These are provisional since not enough calibration data has been collected on phase distributions to date.

The Waterfall phase distribution percentages in Table 45 are numbers from COCOMO 81 used in USC COCOMO II.2000. The percentages vary as product size varies from 2 KSLOC to 512 KSLOC (see the E = 1.12 line in Figures 5 and 6). The values are taken from the COCOMO 81 Semidetached (average) mode provided in Table 6-8 of [Boehm 1981], except for the Transition phase. This phase was undefined in COCOMO 81 and is set equal to MBASE values in Table 46. The percentages from PRR to SWAR add up to 100%. The percentages for Plans and Requirements and Transition are in addition to the 100% of the effort and schedule quantities estimated by COCOMO II.

Table 45. Waterfall Phase Distribution Percentages

Phase (end points)	Effort%	Schedule%
Plans and Requirements (LCCR-PRR)	7 (2-15)	16-24 (2-30)
Product Design (PRR-PDR)	17	24-28
Programming (PDR-UTC)	64-52	56-40
Detailed Design (PDR-CDR)	27-23	
Code and Unit Test (CDR-UTC)	37-29	
Integration and Test (UTC-SWAR)	19-31	20-32

Table 45. Waterfall Phase Distribution Percentages

Phase (end points)	Effort%	Schedule%
Transition (SWAR-SAR)	12 (0-20)	12.5 (0-20)

The MBASE phase distribution percentages in Table 46 are chosen to be consistent with those provided for the Rational RUP in [Royce 1998] and [Kruchten 1999]. They are rescaled to match the COCOMO II definition that 100% of the development effort is done in the Elaboration and Construction phases (between the LCO and IOC milestones, for which most calibration data is available). The corresponding figures for the RUP development cycle are also provided in Table 46.

Table 46. MBASE and RUP Phase Distribution Percentages

	MBASE		RUP	
Phase (end points)	Effort%	Schedule%	Effort%	Schedule%
Inception (IRR to LCO)	6 (2-15)	12.5 (2-30)	5	10
Elaboration (LCO to LCA)	24 (20-28)	37.5 (33-42)	20	30
Construction (LCA to IOC)	76 (72-80)	62.5 (58-67)	65	50
Transition (IOC to PRR)	12 (0-20)	12.5 (0-20)	10	10
Totals:	118	125	100	100

6.3.1 Variations in Effort and Schedule Distributions

The effort and schedule distributions in the Waterfall model vary somewhat by size, primarily reflecting the amount of integration and test required. But the major variations in both the Waterfall model and MBASE/RUP phase effort and schedule quantities come in the phases outside the core development phases (Plans & Requirements and Transition for Waterfall; Inception and Transition for MBASE/RUP).

These large variations are the main reason that the main COCOMO II development estimates do not cover these outer phases (the other strong reason is that calibration data is scanty for the outer phases).

At this time, there is no convenient algorithm for determining whether your Inception phase effort will be nearer to 2% than 15% and Inception phase schedule will be nearer to 2% than 30% of the COCOMO II development cost estimates. The best we can offer at this time is Table 47, which identifies the primary effort and schedule drivers for the Inception and Transition phases.

These are presented in descending order of their effect on Inception phase effort and schedule, and as well as possible in ascending order of their corresponding effect on the Transition phase.

Table 47. Inception and Transition Phase Effort and Schedule Drivers

Factor	Inception	Transition
1. Complexity of LCO issues needing resolution	Very Large	Small
2. System involves major changes in stakeholder roles and responsibilities	Very Large	Large
3. Technical risk level	Large	Some
4. Stakeholder trust level	Large	Considerable
5. Heterogeneous stakeholder communities: Expertise, task nature, language, culture, infrastructure	Large	Large
6. Hardware/software integration	Large	Large
7 Complexity of transition from legacy system	Considerable	Large
8. Number of different installations, classes of installation	Some	Very Large

Note: Order of ratings – Small, Some, Considerable, Large, Very Large

For example, on Factor 1, the stakeholders might enter the Inception phase with a very strong consensus that they wish to migrate some well-defined existing capabilities to a highly feasible client-server architecture. In this case, one could satisfy the LCO criteria with roughly 2% each of the development effort and schedule. On the other hand, if the stakeholders entered the Inception phase with strongly conflicting positions on desired capabilities, priorities, infrastructure, etc., it could take up to 15% of the development effort and 30% of the development schedule to converge to a stakeholder-consensus LCO package (a very large effect).

However, these differences would have a relatively small effect on the amount of effort and schedule it would take to transition the system as defined by the LCO. So the baseline Transition phase percentages of 12% added effort and 12.5% added schedule would be reasonable initial values to use.

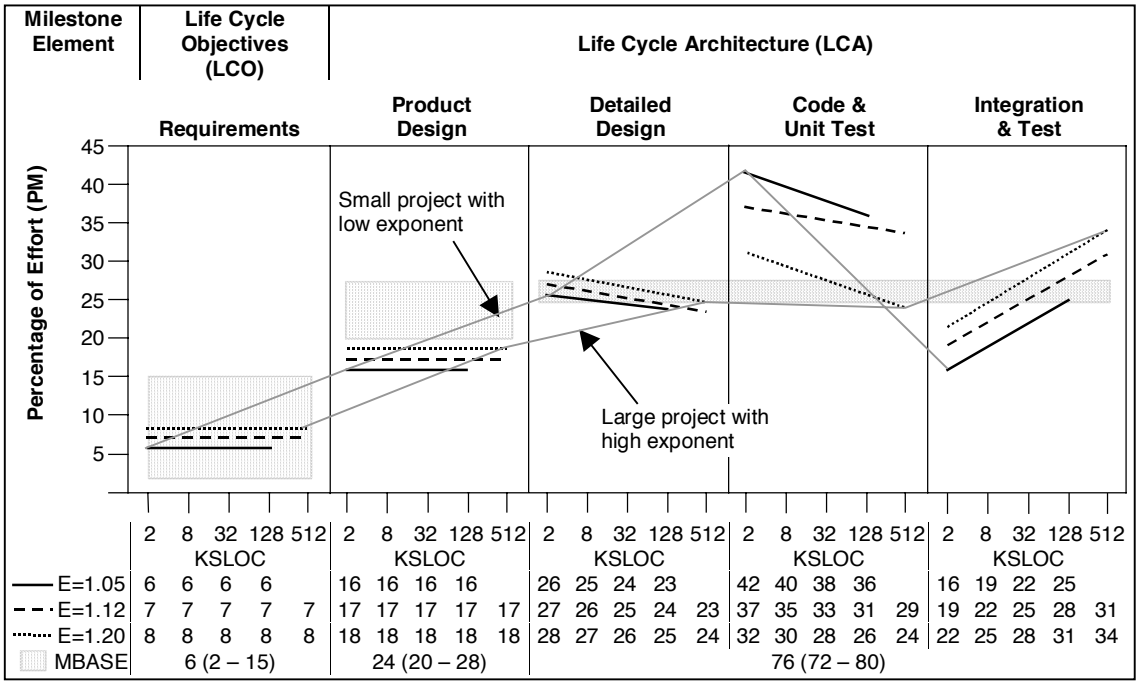
Some of the Inception issues might persist into the Elaboration phase; such persisting issues are the main source of the variations in relative effort and schedule between the Elaboration and Construction phases shown in Table 46. Thus, if you estimate 30% added Inception schedule to achieve a difficult LCO consensus, you may wish to adjust the Elaboration schedule upward from 37.5% to something like 42%. Factors like your COCOMO II TEAM rating would provide additional total effort and schedule to divide between Elaboration and Construction.

In some cases, a factor can have a strong effect on both the Inception and Transition phases. Factor 2 is an example: if the system's effects involve changes in stakeholder roles and responsibilities (e.g., turf, control, power), the amount of effort and schedule will be increased significantly both in negotiating the changes and implementing them.

Some additional sources of variation in phase distributions are deferred for later versions of COCOMO II. These include phase-dependent effort multipliers (as in Detailed COCOMO 81); effects of language level (reduced Construction effort for very high level languages); and effects of optimizing one's project on development cost, schedule, or quality.

6.3.2 Distribution of Effort Across Life Cycle Phases

Figure 5 shows the Waterfall and MBASE phases for distribution of estimated effort. The Waterfall phase distributions are adapted from those in [Boehm 1981; Table 6-8]. The figure shows that distribution of effort varies by size of the product and the size exponent, E.

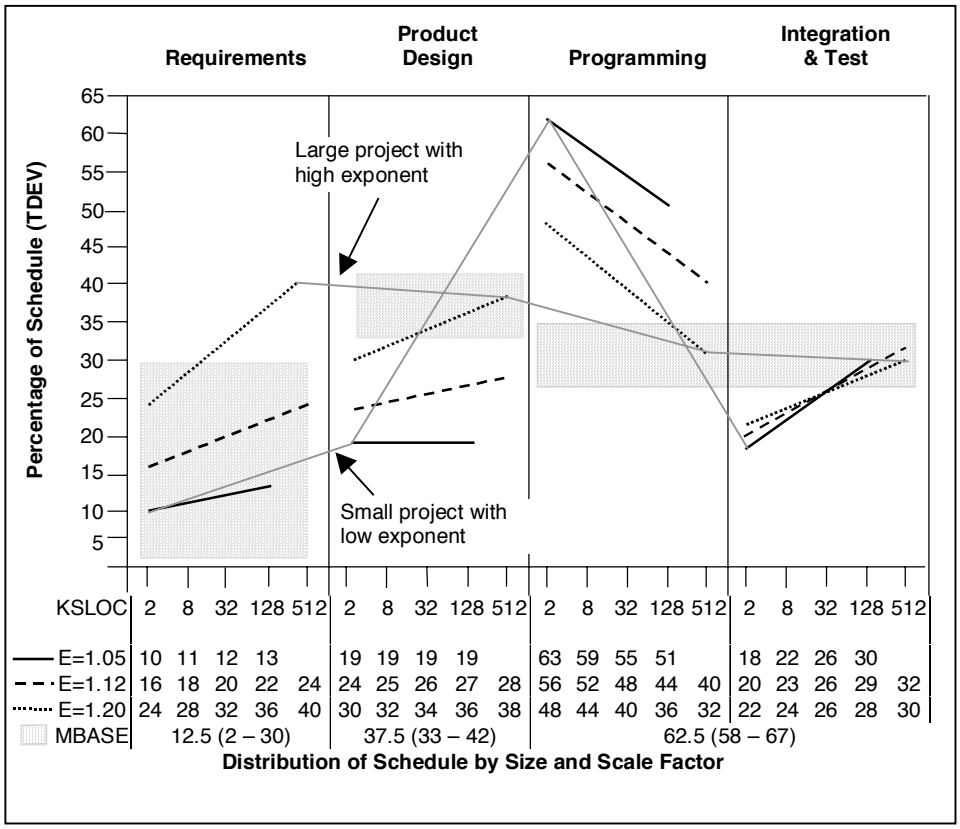


The size exponent E corresponds to the three modes in COCOMO 81. Note that the effort distribution for a small project with a low value for E has the most effort in the Code and Unit Test phase. The top line shows this condition. A large project with a value of E has the most effort concentrated in the Integration and Test phase. This is shown by the bottom line. These distributions of effort are for a Waterfall model project where the development is done in a single sequence through the phases.

Figure 5. Effort Distribution

The MBASE distribution of effort is taken from Table 46. The distribution of effort in the table is 72 to 80 % for the Construction phase which includes Detailed Design, Code and Unit Test, and Integration and Test. The shaded areas in Figure 5 are approximations of the distribution of the Construction effort.

Contrast the Waterfall distributions with the MBASE/RUP distributions. MBASE/RUP emphasizes planning up front and smaller, repeated iterations to develop the product. This makes the distribution of effort less dependent on size and scale factors. The iterative approach also starts the product integration earlier reducing large-system integration gridlock that can occur if integration is left till the last step (as in the Waterfall model).



6.3.3 Distribution of Schedule Across Life Cycle Phases

Figure 6 shows the Waterfall and MBASE phase distribution of estimated schedule. The waterfall distributions are taken from [Boehm 1981; Table 6-8]. As with effort discussed above, schedule varies with size and the scale factor. The two lines bound the range of Waterfall schedule distribution showing a small easy project and a large difficult project.

The MBASE schedule distribution is taken from Table 46. The shaded areas show the range of distribution for each phase.

Figure 6. Distribution of Schedule

6.4 Waterfall and MBASE/RUP Activity Definitions

6.4.1 Waterfall Model Activity Categories

The COCOMO II Waterfall model estimates effort for the following eight major activities [Boehm 1981; Table 4-2]:

- *Requirements Analysis*: Determination, specification, review, and update of software functional, performance, interface, and verification requirements.
- *Product Design*: Determination, specification, review, and update of hardware-software architecture, program design, and database design.

- *Programming*: Detailed design, code, unit test, and integration of individual computer program components. Includes programming personnel planning, tool acquisition, database development, component-level documentation, and intermediate level programming management.
- *Test Planning*: Specification, review, and update of product test and acceptance test plans. Acquisition of associated test drivers, test tools, and test data.
- *Verification and Validation*: Performance of independent requirements validation, design V & V, product test, and acceptance test. Acquisition of requirements and design V & V tools.
- *Project Office Functions*: Project-level management functions. Includes project-level planning and control, contract and subcontract management, and customer interface.
- *Configuration Management and Quality Assurance*: Configuration management includes product identification, change control, status accounting, operation of program support library, development and monitoring of end item acceptance plan. Quality assurance includes development and monitoring of project standards, and technical audits of software products and processes.
- *Manuals*: Development and update of users' manuals, operators' manuals, and maintenance manuals.

When the COCOMO II model is used to estimate effort, the estimated effort can be distributed across the major activities. Section 6.5 provides the percentage distributions of activity within each phase.

6.4.2 Waterfall Model Work Breakdown Structure

The COCOMO II Waterfall and MBASE/RUP activity distributions are defined in more detail via work breakdown structures. Table 48 shows a work breakdown structure outline adapted from [Boehm 1981; Figure 4-6B]. This WBS excludes the requirements-related activities done up to SRR.

Table 48. Software Activity Work Breakdown Structure

1. Management
1.1. Cost, schedule, performance management
1.2. Contract management
1.3. Subcontract management
1.4. Customer interface
1.5. Branch office management
1.6. Management reviews and audits

Table 48. Software Activity Work Breakdown Structure

2. System Engineering 2.1. Software Requirements 2.1.1. Requirements update 2.2. Software product design 2.2.1. Design 2.2.2. Design V & V 2.2.3. Preliminary design review 2.2.4. Design update 2.2.5. Design tools 2.3. Configuration management 2.3.1. Program support library 2.4. End item acceptance 2.5. Quality assurance 2.5.1. Standards
3. Programming 3.1. Detailed design 3.2. Code and unit test 3.3. Integration
4. Test and Evaluation 4.1. Product test 4.1.1. Plans 4.1.2. Procedures 4.1.3. Test 4.1.4. Reports 4.2. Acceptance test 4.2.1. Plans 4.2.2. Procedures 4.2.3. Test 4.2.4. Reports 4.3. Test support 4.3.1. Test beds 4.3.2. Test tools 4.3.3. Test data
5. Data 5.1. Manuals

6.4.3 MBASE/RUP Model Activity Categories

6.4.3.1 Background

This section defines phase and activity distribution estimators for projects using the life-cycle model provided by USC's Model-Based (System) Architecting and Software Engineering (MBASE) approach and the Rational Unified Process (RUP). Both MBASE and RUP use the same phase definitions and milestones. MBASE has a more explicit emphasis on a stakeholder win-win approach to requirements determination and management. RUP accommodates such an approach but more as an option.

In developing these estimators, we have tried to maintain strong consistency with the published Rational phase and activity distributions in [Royce 1998; Kruchten 1999; Jacobson et

al. 1999], and with the published anchor point/MBASE phase boundary definitions in [Boehm 1996, Boehm-Port 1999]. We have iterated and merged drafts of the estimators and definitions with Rational.

Our main sources of information on the Rational phase and activity distributions are:

The common table of default estimators of effort and schedule distribution percentages by phase on page 148 of Royce, page 118 of Kruchten, and page 335 of Jacobson *et al*.

The default estimators of total project activity distribution percentages in Table 10-1, page 148 of Royce. These in turn draw on the definitions of the activity categories in Royce's Life Cycle Phase Emphases (Table 8-1, page 120) and Default Work Breakdown Structure -- WBS-- (Figure 10-2, pages 144-145).

We and Rational agree that these and the COCOMO table values below cannot fit all project situations, and should be considered as draft values to be adjusted via context and judgement to fit individual projects. We have research activities underway to provide stronger guidance on the factors affecting phase and activity distributions.

6.4.3.2 Phase and Activity Category Definitions

Thus, we have begun with the overall phase and activity distributions in [Royce 1998] and used these to develop a set of default MBASE/RUP phase and activity distributions for use in COCOMO II as a counterpart to those provided for the waterfall model. In the process, we found the need to elaborate a few of the activity-category definitions in Royce's Figure 10-2 (e.g., including configuration management within Environment; adding stakeholder coordination as a Management activity and stakeholder requirements negotiation as a Requirements activity; and including explicit Transition Plan and Maintenance Plan activities in Deployment). We also modified a few definitions and allocations (using "evolution" in place of "maintenance;" splitting "Business case development" and "Business case analysis" between Management and Assessment).

For comparison, we have reproduced Royce's WBS as Table 49 and provided the counterpart COCOMO II WBS as Table 50. We have also orthogonalized the WBS organization in Table 50 (e.g., for each phase X, the planning WBS element is AXA and the control element is AXB), and provided more explicit categories corresponding to the Level 2 and 3 project-oriented Key Process Areas in the SEI Capability Maturity model. An exception is Software Quality Assurance, where we agree with Royce that all activities and all people are involved in SQA, and that a separate WBS element for QA is inappropriate.

Table 49. Rational Unified Process Default Work Breakdown Structure [Royce 1998]

A Management AA Inception phase management AAA Business case development AAB Elaboration phase release specifications AAC Elaboration phase WBS* baselining AAD Software development plan AAE Inception phase project control and status assessments AB Elaboration phase management ABA Construction phase release specifications ABB Construction phase WBS baselining ABC Elaboration phase project control and status assessments AC Construction phase management ACA Deployment phase planning ACB Deployment phase WBS baselining ACC Construction phase project control and status assessments AD Transition phase management ADA Next generation planning ADB Transition phase project control and status assessments
B Environment BA Inception phase environment specification BB Elaboration phase environment baselining BBA Development environment installation and administration BBB Development environment integration and custom toolsmithing BBC SCO* database formulation BC Construction phase environment maintenance BCA Development environment installation and administration BCB SCO database maintenance BD Transition phase environment maintenance BDA Development environment maintenance and administration BDB SCO database maintenance BDC Maintenance environment packaging and transition
C Requirements CA Inception phase requirements development CAA Vision specification CAB Use case modeling CB Elaboration phase requirements baselining CBA Vision baselining CBB Use case model baselining CC Construction phase requirements maintenance CD Transition phase requirements maintenance
D Design DA Inception phase architecture prototyping DB Elaboration phase architecture baselining DBA Architecture design modeling DBB Design demonstration planning and conduct DBC Software architecture description DC Construction phase design modeling DCA Architecture design model maintenance DCB Component design modeling DD Transition phase design maintenance

Table 49. Rational Unified Process Default Work Breakdown Structure [Royce 1998]

E Implementation EA Inception phase component prototyping EB Elaboration phase component implementation EBA Critical component coding demonstration integration EC Construction phase component implementation ECA Initial release(s) component coding and stand-alone testing ECB Alpha release component coding and stand-alone testing ECC Beta release component coding and stand-alone testing ECD Component maintenance ED Transition phase component maintenance
F Assessment FA Inception phase assessment planning FB Elaboration phase assessment FBA Test modeling FBB Architecture test scenario implementation FBC Demonstration assessment and release descriptions FC Construction phase assessment FCA Initial release assessment and release description FCB Alpha release assessment and release description FCC Beta release assessment and release description FD Transition phase assessment FDA product release assessment and release descriptions
G Deployment GA Inception phase deployment planning GB Elaboration phase deployment planning GC Construction phase deployment GCA User manual baselining GD Transition phase deployment GDA Product transition to user

*Acronyms: SCO – Software Change Order
WBS – Work Breakdown Structure

Table 50. COCOMO II MBASE/RUP Default Work Breakdown Structure

A Management AA Inception phase management AAA Top-level Life Cycle Plan (LCO* version of LCP*) AAB Inception phase project control and status assessments AAC Inception phase stakeholder coordination and business case development AAD Elaboration phase commitment package and review (LCO package preparation and ARB* review) AB Elaboration phase management ABA Updated LCP with detailed Construction plan (LCA* version of LCP) ABB Elaboration phase project control and status assessments ABC Elaboration phase stakeholder coordination and business case update ABD Construction phase commitment package and review (LCA package preparation and ARB review) AC Construction phase management ACA Updated LCP with detailed Transition and Maintenance plans ACB Construction phase project control and status assessments ACC Construction phase stakeholder coordination ACD Transition phase commitment package and review (IOC* package preparation and PRB* review) AD Transition phase management ADA Updated LCP with detailed next-generation planning ADB Transition phase project control and status assessments ADC Transition phase stakeholder coordination ADD Maintenance phase commitment package and review (PRR* package preparation and PRB review)
B Environment and Configuration Management (CM) BA Inception phase environment/CM scoping and initialization BB Elaboration phase environment/CM BBA Development environment installation and administration BBB Elaboration phase CM BBC Development environment integration and custom toolsmithing BC Construction phase environment/CM evolution BCA Construction phase environment evolution BCB Construction phase CM BD Transition phase environment/CM evolution BDA Construction phase environment evolution BDB Transition phase CM BDC Maintenance phase environment packaging and transition
C Requirements CA Inception phase requirements development CAA Operational Concept Description and business modeling (LCO version of OCD*) CAB Top-level System and Software Requirements Definition (LCO version of SSRD*) CAC Initial stakeholder requirements negotiation CB Elaboration phase requirements baselining CBA OCD elaboration and baselining (LCA version of OCD) CBB SSRD elaboration and baselining (LCA version of SSRD) CC Construction phase requirements evolution CD Transition phase requirements evolution

Table 50. COCOMO II MBASE/RUP Default Work Breakdown Structure

D Design DA Inception phase architecting DAA Top-level System and Software Architecture Description (LCD version of SSAD*) DAB Evaluation of candidate COTS* components DB Elaboration phase architecture baselining DBA SSAD elaboration and baselining DBB COTS integration assurance and baselining DC Construction phase design DCA SSAD evolution DCB COTS integration evolution DCC Component design DD Transition phase design evolution
E Implementation EA Inception phase prototyping EB Elaboration phase component implementation EBA Critical component implementation EC Construction phase component implementation ECA Alpha release component coding and stand-alone testing ECB Beta release (IOC) component coding and stand-alone testing ECC Component evolution ED Transition phase component evolution
F Assessment FA Inception phase assessment FAA Initial assessment plan (LCO version; part of SDP*) FAB Initial Feasibility Rationale Description (LCO version of FRD*) FAC Inception phase element-level inspections and peer reviews FAD Business case analysis (part of FRD) FB Elaboration phase assessment FBA Elaboration of assessment plan (LCA version; part of SDP) FBB Elaboration of feasibility rationale (LCA version of FRD) FBC Elaboration phase element-level inspections and peer reviews FBD Business case analysis update FC Construction phase assessment FCA Detailed test plans and procedures FCB Evolution of feasibility rationale FCC Construction phase element-level inspections and peer reviews FCD Alpha release assessment FCE Beta release (IOC) assessment FD Transition phase assessment
G Deployment GA Inception phase deployment planning (LCO version; part of SDP) GB Elaboration phase deployment planning (LCA version; part of SDP) GC Construction phase deployment planning and preparation GCA Transition plan development GCB Evolution plan development GCC Transition preparation GD Transition phase deployment

Table 50. COCOMO II MBASE/RUP Default Work Breakdown Structure

*Acronyms: ARB -- Architecture Review Board
CM -- Configuration Management
COTS -- Commercial-Off-The-Shelf
FRD -- Feasibility Rationale Description
IOC -- Initial Operational Capability milestone
LCA -- Life Cycle Architecture milestone
LCO -- Life Cycle Objectives milestone
LCP -- Life Cycle Plan
OCD -- Operational Concept Description
PRR -- Product Release Review milestone
PRB -- Product Review Board
SSAD -- System and Software Architecture Description
SSRD -- System and Software Requirements Definition

6.5 Distribution of Effort Across Activities

6.5.1 Waterfall Model Activity Distribution

Tables 51 through 54, adapted from [Boehm 1981; Tables 7-1, 7-2, 7-3], show the distribution of eight major activities (discussed in Section 6.4.1) across the estimated project effort per phase. The activity distribution values should be interpolated for projects that are between values in the table.

For example, a project with size 128 KSLOC and an exponent of 1.12 would spend 28% of its effort in Integration and Test, see Table 54 Overall Phase Percentage row. From Table 54 we see that the Requirement Analysis activity takes 2.5% of the 28%, Product Design takes 5% of the 28%, Programming takes 39% of the 28%, and so on. We see that the activity tables break up the estimated effort for a phase into the different activities that occur during the phase.

Table 51. Plans and Requirements Activity Distribution

	Size Exponent													
	E = 1.05				E = 1.12					E = 1.20				
Size:	**S**	**I**	**M**	**L**	**S**	**I**	**M**	**L**	**VL**	**S**	**I**	**M**	**L**	**VL**
Overall Phase Percentage			6		7	7	7	7	7	8	8	8	8	8
Requirements Analysis			46		48	47	46	45	44	50	48	46	44	42
Product Design			20		16	16.5	17	17.5	18	12	13	14	15	16
Programming			3		2.5	3.5	4.5	5.5	6.5	2	4	6	8	10
Test Planning			3		2.5	3	3.5	4	4.5	2	3	4	5	6
V&V			6		6	6.5	7	7.5	8	6	7	8	9	10
Project Office			15		15.5	14.5	13.5	12.5	11.5	16	14	12	10	8
CM/QA			2		3.5	3	3	3	2.5	5	4	4	4	3
Manuals			5		6	6	5.5	5	5	7	7	6	5	5

S: 2 KSLOC; I: 8 KSLOC; M: 32 KSLOC; L: 128 KSLOC; VL: 512 KSLOC

Table 52. Product Design Activity Distribution

	Size Exponent													
	E = 1.05				E = 1.12					E = 1.20				
Size:	**S**	**I**	**M**	**L**	**S**	**I**	**M**	**L**	**VL**	**S**	**I**	**M**	**L**	**VL**
Overall Phase Percentage	16				17	17	17	17	17	18	18	18	18	18
Requirements Analysis	15				12.5	12.5	12.5	12.5	12.5	10	10	10	10	10
Product Design	40				41	41	41	41	41	42	42	42	42	42
Programming	14				12	12.5	13	13.5	14	10	11	12	13	14
Test Planning	5				4.5	5	5.5	6	6.5	4	5	6	7	8
V&V	6				6	6.5	7	7.5	8	6	7	8	9	10
Project Office	11				13	12	11	10	9	15	13	11	9	7
CM/QA	2				3	2.5	2.5	2.5	2	4	3	3	3	2
Manuals	7				8	8	7.5	7	7	9	9	8	7	7

S: 2 KSLOC; I: 8 KSLOC; M: 32 KSLOC; L: 128 KSLOC; VL: 512 KSLOC

Table 53. Programming Activity Distribution

	Size Exponent													
	E = 1.05				E = 1.12					E = 1.20				
Size:	**S**	**I**	**M**	**L**	**S**	**I**	**M**	**L**	**VL**	**S**	**I**	**M**	**L**	**VL**
Overall Phase Percentage	68	65	62	59	64	61	58	55	52	60	57	54	51	48
Requirements Analysis	5				4	4	4	4	4	3	3	3	3	3
Product Design	10				8	8	8	8	8	6	6	6	6	6
Programming	58				56.5	56.5	56.5	56.5	56.5	55	55	55	55	55
Test Planning	4				4	4.5	5	5.5	6	4	5	6	7	8
V&V	6				7	7.5	8	8.5	9	8	9	10	11	12
Project Office	6				7.5	7	6.5	6	5.5	9	8	7	6	5
CM/QA	6				7	6.5	6.5	6.5	6	8	7	7	7	6
Manuals	5				6	6	5.5	5	5	7	7	6	5	5

S: 2 KSLOC; I: 8 KSLOC; M: 32 KSLOC; L: 128 KSLOC; VL: 512 KSLOC

Table 54. Integration and Test Activity Distribution

	Size Exponent													
	E = 1.05				E = 1.12					E = 1.20				
Size:	**S**	**I**	**M**	**L**	**S**	**I**	**M**	**L**	**VL**	**S**	**I**	**M**	**L**	**VL**
Overall Phase Percentage	16	19	22	25	19	22	25	28	31	22	25	28	31	34
Requirements Analysis	3				2.5	2.5	2.5	2.5	2.5	2	2	2	2	2
Product Design	6				5	5	5	5	5	4	4	4	4	4
Programming	34				33	35	37	39	41	32	36	40	44	48
Test Planning	2				2.5	2.5	3	3	3.5	3	3	4	4	5

Table 54. Integration and Test Activity Distribution

	Size Exponent													
	E = 1.05				E = 1.12					E = 1.20				
Size:	S	I	M	L	S	I	M	L	VL	S	I	M	L	VL
V&V	34				32	31	29.5	28.5	27	30	28	25	23	20
Project Office	7				8.5	8	7.5	7	6.5	10	9	8	7	6
CM/QA	7				8.5	8	8	8	7.5	10	9	9	9	8
Manuals	7				8	8	7.5	7	7	9	9	8	7	7

S: 2 KSLOC; I: 8 KSLOC; M: 32 KSLOC; L: 128 KSLOC; VL: 512 KSLOC

The last two activity tables handle the situation where development or maintenance is performed as a level of effort, Tables 55 and 56 respectively. In other words, there is a fixed amount of staff that will be working on the project and the eight major activities are divided among the fixed staffing.

As an example, a maintenance project has 10 staff for 12 months (120 Person-Months), a size of 8 KSLOC, and an exponent of 1.20. From Table 56 we see that Requirements Analysis will consume 6% of the staff's effort or 7.2 PM, Program Design will consume 11% of the staff's effort or 13.2 PM, Programming will consume 39% of the staff's effort or 46.8 PM, and so on.

Table 55. Development Activity Distribution

	Size Exponent													
	E = 1.05				E = 1.12					E = 1.20				
Size:	S	I	M	L	S	I	M	L	VL	S	I	M	L	VL
Requirements Analysis	6				5	5	5	5	5	4	4	4	4	4
Product Design	14				13	13	13	13	13	12	12	12	12	12
Programming	48	47	46	45	45	45	44.5	44.5	44.5	42	43	43	44	45
Test Planning	4				4	4	4.5	5	5.5	4	4	5	6	7
V&V	10	11	12	13	11	12	13	13.5	14	12	13	14	14	14
Project Office	7				8.5	8	7.5	7	6.5	10	9	8	7	6
CM/QA	5				6.5	6	6	6	5.5	8	7	7	7	6
Manuals	6				7	7	6.6	6	6	8	8	7	6	6

S: 2 KSLOC; I: 8 KSLOC; M: 32 KSLOC; L: 128 KSLOC; VL: 512 KSLOC

Table 56. Maintenance Activity Distribution

	Size Exponent													
	E = 1.05				E = 1.12					E = 1.20				
Size:	S	I	M	L	S	I	M	L	VL	S	I	M	L	VL
Requirements Analysis	7				6.5	6.5	6.5	6	6	6	6	6	5	5
Product Design	13				12	12	12	12	12	11	11	11	11	11
Programming	45	44	43	42	41.5	41.5	41	41	41	38	39	39	40	41
Test Planning	3				3	3	3.5	4	4.5	3	3	4	5	6

Table 56. Maintenance Activity Distribution

Size:	Size Exponent E = 1.05 S	I	M	L	E = 1.12 S	I	M	L	VL	E = 1.20 S	I	M	L	VL
V&V	10	11	12	13	11	12	13	13.5	14	12	13	14	14	14
Proect Office	7				8.5	8	7.5	7	6.5	10	9	8	7	6
CM/QA	5				6.5	6	6	6	5.5	8	7	7	7	6
Manuals	10				11	11	10.5	10.5	10.5	12	12	11	11	11

S: 2 KSLOC; I: 8 KSLOC; M: 32 KSLOC; L: 128 KSLOC; VL: 512 KSLOC

6.5.2 MBASE/RUP Model Activity Distribution Values

Table 57 shows the resulting COCOMO II MBASE/RUP default phase and activity distribution values. The first two lines show the Rational and COCOMO II schedule percentages by phase. Their only difference is that the Rational percentages sum to 100% for the full set of Inception, Elaboration, Construction, and Transition phases (IECT), while COCOMO II counts the core Elaboration and Construction phases as 100%. This is done because the scope and duration of the Inception and Transition phases are much more variable, and because less data is available to calibrate estimation models for these phases. For COCOMO II, this means that the current model covering the Elaboration and Construction phases is considerably more accurate and robust than we could achieve with counterpart models that would include the Inception and/or Transition phases.

Lines 1 and 2 in Table 57 show the Rational and COCOMO II phase distributions for the project's schedule in months. Lines 3 and 4 in Table 57 show the corresponding phase distributions for effort. The sum of the COCOMO II percentages for the total IECT project span is 125% for schedule and 118% for effort.

Table 57. COCOMO II MBASE/RUP Phase and Activity Distribution Values

		Development			Total		Total
	Inception	Elaboration	Construction	Transition	Royce	COCOMO II	Maintenance
Rational Schedule	10	30	50	10	100		
COCOMO II Schedule	12.5	37.5	62.5	12.5		125	
Rational Effort	5	20	65	10	100		
COCOMO II Effort	6	24	76	12		118	100
Activity % of phase / IECT	100	100	100	100	118	118	100
Management	14	12	10	14	12	13	11
Environment / CM	10	8	5	5	12	7	6
Requirements	38	18	8	4	12	13	12
Design	19	36	16	4	18	22	17
Implementation	8	13	34	19	29	32	24
Assessment	8	10	24	24	29	24	22

Table 57. COCOMO II MBASE/RUP Phase and Activity Distribution Values

		Development			Total		Total
	Inception	Elaboration	Construction	Transition	Royce	COCOMO II	Maintenance
Deployment	3	3	3	30	6	7	8

Lines 6 to 12 in Table 57 show the default percentage of effort by activity for each of the MBASE/RUP phases. For example, the Management activities are estimated to consume 14% of the effort in the Inception phase, 12% in the Elaboration phase, 10% in Construction, and 14% in Transition. For the total IECT span, the Management activities consume

(14%)(6%) + (12%)(24%) + (10%)(76%) + (14%)(12%) = 13%

This sum is slightly larger but quite comparable to the IECT percentage of 12% derived from Royce's Table 10-1. The WBS definitions in Tables 49 and 50 are sufficiently similar that the values in Table 57 can be applied equally well to both.

These values can be used to determine draft project staffing plans for each of the phases. For example, suppose there was a 100 KSLOC software system that had an estimated development effort of 466 PM and an estimated schedule of 26 months. From lines 2 and 4 of Table 57, we can compute the estimated schedule and effort of the Construction phase as:

Schedule: (26 Mo.) (.625) = 16.25 Mo.

Effort: (466 PM) (.76) = 354 PM

The average staff level of the Construction phase is thus:

354 PM/ 16.25 Mo. = 21.8 persons.

We can then use lines 6-12 of Table 57 to provide a draft estimate of what these 21.8 persons will be doing during the Construction phase. For example, the estimated average number of personnel performing Management activities is:

(21.8 persons) (.10) = 2.2 persons

Table 58 shows the full set of draft activity estimates for the Construction phase.

Table 58. Example Staffing Estimate for Construction Phase

Activity	Percentage	Average Staff
Total	100	21.8
Management	10	2.2
Environment	5	1.1
Requirements	8	1.7
Design	16	3.5
Implementation	34	7.4
Assessment	24	5.2

Table 58. Example Staffing Estimate for Construction Phase

Activity	Percentage	Average Staff
Deployment	3	0.7

These staffing estimates can be used for other purposes as well. When multiplied by associated average labor costs for activity categories, they can be used as starting points for project WBS and budget allocations, or for earned values associated with phase deliverables.

It is important to understand that these numbers are just draft default starting points for the actual numbers you use to manage your project. Every project will have special circumstances which should be considered in adjusting the draft values (see also [Royce 1998; p. 218; Kruchten 1999; pp.118-119; Jacobson et al. 1999; p. 336]). For example, an ultra-reliable product will have higher Assessment efforts and costs; a project with a stable environment already in place will have lower up-front Environment efforts and costs. The COCOMO II research agenda includes activities to provide further guidelines for adjusting the phase and activity distributions to special circumstances. Even then, however, the estimated phase and activity distribution numbers should be subject to critical review. As with other COCOMO II estimates, the default phase and activity distribution estimates should be considered as a stimulus to thought, and not as a substitute for thought.

6.6 Definitions and Assumptions

COCOMO II's definitions and assumptions are similar to those for COCOMO 81 [Boehm 1981; pp. 58-61], but with some differences. Here is a summary of similarities and differences.

1. Sizing. COCOMO 81 just used Delivered Source Instructions for sizing. COCOMO II uses combinations of Function Points (FP) and Source Lines of Code for the Early Design and Post-Architecture models, with counting rules in [IFPUG 1994] for FP and Table 63 for SLOC.
2. Development Periods Included. For the Waterfall process model, COCOMO II uses the same milestone end points (Software Requirements Review to Software Acceptance Review) as COCOMO 81. For the MBASE/RUP process model, COCOMO II uses the Life Cycle Objectives and Initial Operational Capability milestone as end points for counting effort and schedule. Details for both are in Section 6.2.
3. Project Activities Included. For the Waterfall process model, COCOMO II includes the same activities as did COCOMO 81. For the MBASE and RUP process models, the Work Breakdown Structures in Section 6.4 define the project activities included by phase. For all the models, all software development activities such as documentation, planning and control, and configuration management (CM) are included, while database administration is not. For all the models, the software portions of a hardware-software project are included (e.g., software CM, software project management) but general CM and management are not. Both models have add-on efforts for a front-end phase (Plans and Requirements for COCOMO 81; Inception for (MBASE/RUP). COCOMO II differs from COCOMO 81 in having

add-on efforts for a back-end Transition phase, including conversion, installation, and training. As discussed in Section 6.3 the size of these add-on efforts can vary a great deal, and their effort estimates should be adjusted for particularly small or large add-on endeavors.

4. Labor categories included. COCOMO 81 and COCOMO II estimates both use the same definitions of labor categories included as direct-charged project effort vs. overhead effort. Thus, they include project managers and program librarians, but exclude computer center operators, personnel-department personnel, secretaries, higher management, janitors, and so on.
5. Dollar Estimates. COCOMO 81 and COCOMO II avoid estimating labor costs in dollars because of the large variations between organizations in what is included in labor costs, e.g. unburdened (by overhead cost), burdened, including pension plans, office rental, and profit margin. Person months are a more stable quantity than dollars given current inflation rates and international money fluctuations.
6. Person-month definition. A COCOMO PM consists of 152 hours of working time. This has been found to be consistent with practical experience with the average monthly time off because of holidays, vacation, and sick leave. To convert a COCOMO estimate in person-months to other units, use the following:

Person-hours: multiply by 152
Person-days: multiply by 19
Person-years: divide by 12

Some implementations, such as USC COCOMO II, provide an adjustable parameter for the number of person-hours per person-month. Thus, if you enter 137 (10% less) for this parameter, USC COCOMO II will increase your person-month estimate by 10%, and calculate an appropriately longer development schedule.

7. Model Calibration to the Local Environment

Studies of the data used to calibrate all the parameters of the COCOMO II Post-Architecture model have shown the model to be significantly more accurate when calibrated to an organization. All that was done in the study was to calibrate the constant, A, in the effort estimation equation. This is simple enough to do that it can be performed with a calculator, spreadsheet, or a statistical regression tool.

The intent of calibration is to take the productivity and activity distributions of the local development environment into account. Figure 7 is a fictitious example of data from 8 projects. The effect of calibrating the constant A is to raise or lower the fitted line from the "out-of-the-box" COCOMO estimates to estimates that reflect local conditions.

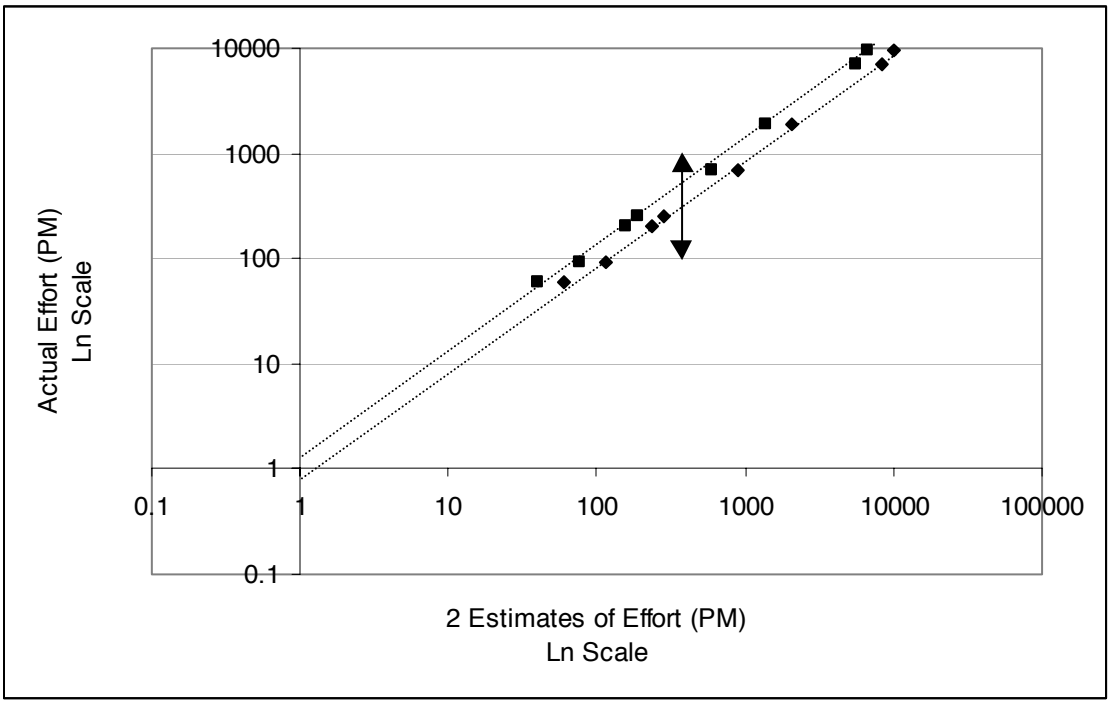


Figure 7. Difference Between COCOMO II and Local Calibrations

Calibration to the local environment consists of adjusting the constant, A, in the model. Only the calibration of A is discussed here; however for calibration of A and E see Chapter 4 in [Boehm et al. 2000]. The applicable portion of Equation 1 is repeated here for this discussion.

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

There is more than one method to calibrate the constant A (see other examples in [Boehm 1981; Chapter 29; Boehm et al. 2000; Chapter 4]. The technique described here uses natural logs. It is recommended that at least 5 data points from projects be used in calibrating the constant A.

As an example, the table below shows eight projects. The data needed is the actual effort (PM_{actual}) that was expended between the end of Requirements Analysis and the end of software Integration and Test. The activities should be the same as those discussed in Section 6.4. The end-product size, Scale Factors, and Cost Drivers are also needed. An unadjusted estimated is created using Equation 1 without the constant A. Next, natural logs (ln) are taken of the actual effort and unadjusted estimate. For each project, the difference between the log of the actual effort and the log of the unadjusted estimate are determined. The average of the differences, X, will determine the constant A by taking the anti-log of the average: $A = e^X$.

Table 59. Example of Local Calibration of A

PMactual	KSLOC	ΠEM_i	E	Unadjusted Estimate	$\ln(PM_{actual})$	ln(Unadjusted Estimate)	Difference
1854.6	134.5	1.89	1.20	686.7	7.53	6.53	0.99
258.5	132.0	0.49	1.08	94.3	5.55	4.55	1.01
201.0	44.0	1.06	1.13	77.7	5.30	4.35	0.95
58.9	3.6	5.05	1.09	20.3	4.08	3.01	1.07
9661.0	380.8	3.05	1.18	3338.8	9.18	8.11	1.06
7021.3	980.0	0.92	1.16	2753.5	8.86	7.92	0.94
91.7	11.2	2.45	1.15	38.9	4.52	3.66	0.86
689.7	61.6	2.38	1.17	301.1	6.54	5.71	0.83
						X=	0.96
						A=	2.62

This example shows that instead of using the COCOMO II.2000 constant of A = 2.94, a local constant of 2.62 should be used for estimating software projects in the local development.

In addition to calibrating the estimation equations, the distribution of the estimates should be calibrated too. Recall the distribution of effort in Section 6.4. This may be different for the local environment due to the use of different development methodologies and organizational processes.

The table below is an example of a collection of effort data by lifecycle phase and by activity. This table could be expanded to cover all activities and phases of the local development lifecycle. Even if the local activities are not all covered by the COCOMO effort estimate the information will still be useful in planning and tracking project progress.

Table 60. Example of Local Effort Data Collection

Activity	Effort by Lifecycle Phase in Person-Months						
	Architectural Design	Software Design	Implementation Build-1 (DD & CUT)	Integration Test Build-1	Implementation Build-2 (DD & CUT)	Integration Test Build-2	Total Effort by Activity
Management	3.72	6.24	7.08	0	9.96	0	27.00
Requirements Engineering	3.72	5.88	4.20	0	7.08	0	20.88
Test Engineering	3.72	7.08	12.24	10.56	11.4	5.16	50.16
Software Engineering (A+B)	19.56	26.88	46.92	3.96	65.28	6.84	169.44
Subsystem A	13.20	16.44	34.56	3.00	42.48	5.76	115.44
Subsystem B	6.36	10.44	12.36	0.96	22.80	1.08	54.00
Support	3.96	4.92	12.00	4.80	10.56	0	36.24
Total Effort by Phase	34.68	51.00	82.44	19.32	104.28	12.00	303.72

The data from the above table can be converted into phase distribution percentages. This is used with the calibrated COCOMO II model to derive estimates broken down by phase and activity.

Table 61. Example of Local Effort Distribution

Activity	Percentage Effort by Lifecycle Phase				
	Architectural Design	Software Design	Implementation	Integration Testing	Total Effort by Activity
Management	1.22%	2.05%	5.61%	0.00%	8.89%
Requirements Engineering	1.22%	1.94%	3.71%	0.00%	6.87%
Test Engineering	1.22%	2.33%	7.78%	5.18%	16.52%
Software Engineering	6.44%	8.85%	36.94%	3.56%	55.79%
Support	1.30%	1.62%	7.43%	1.58%	11.93%
Total Effort by Phase	11.42%	16.79%	61.48%	10.31%	100.00%

Calibration of the model to the local environment is an important activity. The results of the calibration can be an important input to planning and quantitative management practices.

8. Summary

8.1 Models

8.1.1 Sizing equations

The Post-Architecture and Early Design models use the same sizing equations. Sizing is summarized below and discussed in Section 2.

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

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

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

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

Symbol	Description
AA	Assessment and Assimilation
AAF	Adaptation Adjustment Factor
AAM	Adaptation Adjustment Modifier
CM	Percent Code Modified
DM	Percent Design Modified
IM	Percent of Integration Integration Required for the Adapted Software
KSLOC	Thousands of Source Lines of Code
REVL	Requirements Evolution and Volatility
SU	Software Understanding
UNFM	Programmer Unfamiliarity with Software

8.1.2 Post-Architecture Model equations

This model is explained in Section 3.

$$\text{PM} = \text{A} \cdot \text{Size}^{\text{E}} \times \prod_{i=1}^{17} \text{EM}_i + \text{PM}_{\text{Auto}}$$

$$\text{E} = \text{B} + 0.01 \times \sum_{j=1}^{5} \text{SF}_j$$

$$\text{PM}_{\text{Auto}} = \frac{\text{Adapted SLOC} \times \left(\text{AT}/100\right)}{\text{ATPROD}}$$

Symbol	Description
A	Effort coefficient that can be calibrated, currently set to 2.94

AT	Percentage of the code that is re-engineered by automatic translation
ATPROD	Automatic translation productivity
B	Scaling base-exponent that can be calibrated, currently set to 0.91
E	Scaling exponent described in Section 3.1
EM	17 Effort Multipliers discussed in Section 3.2.1
PM	Person-Months effort from developing new and adapted code
PM_{Auto}	Person-Months effort from automatic translation activities discussed in Section 2.6.
SF	5 Scale Factors discussed in Section 3.1

8.1.3 Early Design Model equations

This model is explained in Section 3.

$$PM = A \cdot Size^{E} \times \prod_{i=1}^{7} EM_i + PM_{Auto}$$

Symbol	Description
A	Effort coefficient that can be calibrated, currently set to 2.94
E	Scaling exponent described in Section 3.1
EM	7 Effort Multipliers discussed in Section 3.2.2
PM	Person-Months effort from developing new and adapted code
PM_{Auto}	Person-Months effort from automatic translation activities discussed in Section 2.6.
SF	5 Scale Factors discussed in Section 3.1

8.1.4 Time to Develop equation

This model is explained in Section 4.

$$TDEV = [C \times (PM_{NS})^{F}] \times \frac{SCED\%}{100}$$

$$F = (D + 0.2 \times [E - B])$$

Symbol	Description
B	The scaling base-exponent for the effort equation, currently set to 0.91
C	Coefficient that can be calibrated, currently set to 3.67
D	Scaling base-exponent that can be calibrated, currently set to 0.28
E	The scaling exponent for the effort equation
PM_{NS}	Person-Months estimated without the SCED cost driver (Nominal Schedule)
SCED	Required Schedule Compression
TDEV	Time to Develop in calendar months

8.2 Rating Scales

The rating scales for the scale factors are given below and discussed in Section 3.1.

Scale Factors	Very Low	Low	Nominal	High	Very High	Extra High
PREC	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	thoroughly 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	The estimated Equivalent Process Maturity Level (EPML) or					
	SW-CMM Level 1 Lower	SW-CMM Level 1 Upper	SW-CMM Level 2	SW-CMM Level 3	SW-CMM Level 4	SW-CMM Level 5

The rating scales for the Post-Architecture model cost drivers are given below in Table 62 and discussed in Section 3.2.1. The cost drivers for the Early Design model are discussed in Section 3.2.2

Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
RELY	slight inconvenience	low, easily recoverable losses	moderate, easily recoverable losses	high financial loss	risk to human life	
DATA		Testing DB bytes / Pgm SLOC < 10	10 ≤ D/P < 100	100 ≤ D/P < 1000	D/P > 1000	
CPLX	see Table 19					
RUSE		none	across project	across program	across product line	across multiple product lines
DOCU	Many life-cycle needs uncovered	Some life-cycle needs uncovered.	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very excessive for life-cycle needs	
TIME			≤ 50% use of available execution time	70%	85%	95%
STOR			≤ 50% use of available storage	70%	85%	95%
PVOL		major change every 12 mo.; minor change every 1 mo.	major: 6 mo.; minor: 2 wk.	major: 2 mo.; minor: 1 wk.	major: 2 wk.; minor: 2 days	
ACAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCAP	15th percentile	35th percentile	55th percentile	75th percentile	90th percentile	
PCON	48% / year	24% / year	12% / year	6% / year	3% / year	
APEX	≤ 2 months	6 months	1 year	3 years	6 years	
PLEX	≤ 2 months	6 months	1 year	3 years	6 year	
LTEX	≤ 2 months	6 months	1 year	3 years	6 year	
TOOL	edit, code, debug	simple, frontend, backend CASE, little integration	basic lifecycle tools, moderately integrated	strong, mature lifecycle tools, moderately integrated	strong, mature, proactive lifecycle tools, well integrated with processes, methods, reuse	

Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
SITE: Collo-cation	International	Multi-city **and** multi-company	Multi-city **or** multi-company	Same city or metro area	Same building or complex	Fully collocated
SITE: Com-muni-cation	Some phone, mail	Individual phone, FAX	Narrow-band email	Wide-band electronic communica-tion.	Wide-band elect. comm, occasional video conf.	Interactive multimedia
SCED	75% of nominal	85% of nominal	100% of nominal	130% of nominal	160% of nominal	

8.3 COCOMO II Version Parameter Values

8.3.1 COCOMO II.2000 Calibration

The following table, Table 62, shows the COCOMO II.2000 calibrated values for Post-Architecture scale factors and effort multipliers.

Table 62. COCOMO II 2000 Calibrated Post-Architecture Model Values

Baseline Effort Constants: **A** = 2.94; **B** = 0.91
Baseline Schedule Constants: **C** = 3.67; **D** = 0.28

Driver	Symbol	VL	L	N	H	VH	XH
PREC	SF_1	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	SF_2	5.07	4.05	3.04	2.03	1.01	0.00
RESL	SF_3	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	SF_4	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	SF_5	7.80	6.24	4.68	3.12	1.56	0.00
RELY	EM_1	0.82	0.92	1.00	1.10	1.26	
DATA	EM_2		0.90	1.00	1.14	1.28	
CPLX	EM_3	0.73	0.87	1.00	1.17	1.34	1.74
RUSE	EM_4		0.95	1.00	1.07	1.15	1.24
DOCU	EM_5	0.81	0.91	1.00	1.11	1.23	
TIME	EM_6			1.00	1.11	1.29	1.63
STOR	EM_7			1.00	1.05	1.17	1.46
PVOL	EM_8		0.87	1.00	1.15	1.30	
ACAP	EM_9	1.42	1.19	1.00	0.85	0.71	
PCAP	EM_{10}	1.34	1.15	1.00	0.88	0.76	
PCON	EM_{11}	1.29	1.12	1.00	0.90	0.81	
APEX	EM_{12}	1.22	1.10	1.00	0.88	0.81	
PLEX	EM_{13}	1.19	1.09	1.00	0.91	0.85	
LTEX	EM_{14}	1.20	1.09	1.00	0.91	0.84	
TOOL	EM_{15}	1.17	1.09	1.00	0.90	0.78	
SITE	EM_{16}	1.22	1.09	1.00	0.93	0.86	0.80
SCED	EM_{17}	1.43	1.14	1.00	1.00	1.00	

Table 63 shows the COCOMO II.2000 calibrated values for Early Design effort multipliers. The scale factors are the same as for the Post-Architecture model.

Table 63. COCOMO II.2000 Calibrated Early Design Model Values

Baseline Effort Constants: **A** = 2.94; **B** = 0.91 Baseline Schedule Constants: **C** = 3.67; **D** = 0.28								
Driver	**Symbol**	**XL**	**VL**	**L**	**N**	**H**	**VH**	**XH**
PERS	EM_1	2.12	1.62	1.26	1.00	0.83	0.63	0.50
RCPX	EM_2	0.49	0.60	0.83	1.00	1.33	1.91	2.72
PDIF	EM_3			0.87	1.00	1.29	1.81	2.61
PREX	EM_4	1.59	1.33	1.12	1.00	0.87	0.74	0.62
FCIL	EM_5	1.43	1.30	1.10	1.0	0.87	0.73	0.62
RUSE	EM_6			0.95	1.00	1.07	1.15	1.24
SCED	EM_7		1.43	1.14	1.00	1.00	1.00	

8.3.2 COCOMO II.1997 Calibration

The following table shows the COCOMO II.1997 calibrated values for scale factors and effort multipliers.

Baseline Effort Constants: **A** = 2.45; **B** = 1.01 Baseline Schedule Constants: **C** = 2.66; **D** = 0.33							
Driver	**Symbol**	**VL**	**L**	**N**	**H**	**VH**	**XH**
PREC	SF_1	4.05	3.24	2.43	1.62	0.81	0.00
FLEX	SF_2	6.07	4.86	3.64	2.43	1.21	0.00
RESL	SF_3	4.22	3.38	2.53	1.69	0.84	0.00
TEAM	SF_4	4.94	3.95	2.97	1.98	0.99	0.00
PMAT	SF_5	4.54	3.64	2.73	1.82	0.91	0.00
RELY	EM_1	0.75	0.88	1.00	1.15	1.39	
DATA	EM_2		0.93	1.00	1.09	1.19	
RUSE	EM_3		0.91	1.00	1.14	1.29	1.49
DOCU	EM_4	0.89	0.95	1.00	1.06	1.13	
CPLX	EM_5	0.75	0.88	1.00	1.15	1.30	1.66
TIME	EM_6			1.00	1.11	1.31	1.67
STOR	EM_7			1.00	1.06	1.21	1.57
PVOL	EM_8		0.87	1.00	1.15	1.30	
ACAP	EM_9	1.50	1.22	1.00	0.83	0.67	
PCAP	EM_{10}	1.37	1.16	1.00	0.87	0.74	
PCON	EM_{11}	1.24	1.10	1.00	0.92	0.84	
APEX	EM_{12}	1.22	1.10	1.00	0.89	0.81	
PLEX	EM_{13}	1.25	1.12	1.00	0.88	0.81	
LTEX	EM_{14}	1.22	1.10	1.00	0.91	0.84	
TOOL	EM_{15}	1.24	1.12	1.00	0.86	0.72	
SITE	EM_{16}	1.25	1.10	1.00	0.92	0.84	0.78
SCED	EM_{17}	1.29	1.10	1.00	1.00	1.00	

8.4 Source Code Counting Rules

What is a line of source code? This checklist, adopted from the Software Engineering Institute [Park 1992], attempts to define a *logical* line of source code. The intent is to define a logical line of code while not becoming too language specific for use in collection data to validate the COCOMO 2.0 model.

Table 64. COCOMO II SLOC Checklist

Definition Checklist for Source Statements Counts								
Definition name:	Logical Source Statements (basic definition)				Date:_____ Originator: COCOMO II			
Measurement unit:	**Physical source lines**							
	Logical source statements	√						
Statement type	**Definition**	√	**Data Array**				**Includes**	**Excludes**
When a line or statement contains more than one type, classify it as the type with the highest precedence.								
1 Executable					**Order of precedence:**	1	√	
2 Nonexecutable								
3 Declarations						2	√	
4 Compiler directives						3	√	
5 Comments								
6 On their own lines						4		√
7 On lines with source code						5		√
8 Banners and non-blank spacers						6		√
9 Blank (empty) comments						7		√
10 Blank lines						8		√
How produced	**Definition**	√	**Data array**				**Includes**	**Excludes**
1 Programmed							√	
2 Generated with source code generators								√
3 Converted with automated translators							√	
4 Copied or reused without change							√	
5 Modified							√	
6 Removed								√
Origin	**Definition**	√	**Data array**				**Includes**	**Excludes**
1 New work: no prior existence							√	
2 Prior work: taken or adapted from								
3 A previous version, build, or release							√	
4 Commercial, off-the-shelf software (COTS), other than libraries								√
5 Government furnished software (GFS), other than reuse libraries								√
6 Another product								√
7 A vendor-supplied language support library (unmodified)								√
8 A vendor-supplied operating system or utility (unmodified)								√
9 A local or modified language support library or operating system								√
10 Other commercial library								√
11 A reuse library (software designed for reuse)							√	
12 Other software component or library							√	
Usage	**Definition**	√	**Data array**				**Includes**	**Excludes**

Table 64. COCOMO II SLOC Checklist

Definition Checklist for Source Statements Counts							
Definition name:	Logical Source Statements (basic definition)			Date:_____ Originator: COCOMO II			
1 In or as part of the primary product						√	
2 External to or in support of the primary product							√
Delivery	**Definition**	√	**Data array**			**Includes**	**Excludes**
1 Delivered:							
2 Delivered as source						√	
3 Delivered in compiled or executable form, but not as source							√
4 Not delivered:							
5 Under configuration control							√
6 Not under configuration control							√
Functionality	**Definition**	√	**Data array**			**Includes**	**Excludes**
1 Operative						√	
2 Inoperative (dead, bypassed, unused, unreferenced, or unaccessible):							
3 Functional (intentional dead code, reactivated for special purposes)						√	
4 Nonfunctional (unintentionally present)							√
Replications	**Definition**	√	**Data array**			**Includes**	**Excludes**
1 Master source statements (originals)						√	
2 Physical replicates of master statements, stored in the master code						√	
3 Copies inserted, instantiated, or expanded when compiling or linking							√
4 Postproduction replicates—as in distributed, redundant, or reparameterized systems							√
Development status	**Definition**	√	**Data array**			**Includes**	**Excludes**
Each statement has one and only one status, usually that of its parent unit.							
1Estimated or planned							√
2 Designed							√
3 Coded							√
4 Unit tests completed							√
5 Integrated into components							√
6 Test readiness review completed							√
7 Software (CSCI) tests completed							√
8 System tests completed						√	

Table 64. COCOMO II SLOC Checklist

Definition Checklist for Source Statements Counts								
Definition name:	Logical Source Statements (basic definition)				Date:_____ Originator: COCOMO II			
Language	**Definition**	√	**Data array**				**Includes**	**Excludes**
List each source language on a separate line.								
1 Separate totals for each language							√	
Clarifications	**Definition**	√	**Data array**				**Includes**	**Excludes**
(general)								
1 Nulls, continues, and no-ops							√	
2 Empty statements, e.g. ";;" and lone semicolons on separate lines								√
3 Statements that instantiate generics							√	
4 Begin...end and {...} pairs used as executable statements							√	
5 Begin...end and {...} pairs that delimit (sub)program bodies								√
6 Logical expressions used as test conditions								√
7 Expression evaluations used as subprograms arguments								√
8 End symbols that terminate executable statements								√
9 End symbols that terminate declarations or (sub)program bodies								√
10 Then, else, and otherwise symbols								√
11 Elseif statements							√	
12 Keywords like procedure division, interface, and implementation							√	
13 Labels (branching destinations) on lines by themselves								√
Clarifications	**Definition**	√	**Data array**				**Includes**	**Excludes**
(language specific)								
Ada								
1 End symbols that terminate declarations or (sub)program bodies								√
2 Block statements, e.g. begin...end							√	
3 With and use clauses							√	
4 When (the keyword preceding executable statements)								√
5 Exception (the keyword, used as a frame header)							√	
6 Pragmas							√	
Assembly								
1 Macro calls							√	
2 Macro expansions								√
C and C++								
1 Null statement, e.g. ";" by itself to indicate an empty body								√
2 Expression statements (expressions terminated by semicolons)							√	
3 Expression separated by semicolons, as in a "for" statement							√	
4 Block statements, e.g. {...} with no terminating semicolon							√	
5 ";", ";" or ";" on a line by itself when part of a declaration								√
6 ";" or ";" on a line by itself when part of an executable statement								√
7 Conditionally compiled statements (#if, #ifdef, #ifndef)							√	
8 Preprocessor statements other than #if, #ifdef, and #ifndef							√	
CMS-2								
1 Keywords like SYS-PROC and SYS-DD							√	
COBOL								
1 "PROCEDURE DIVISION", "END DECLARATIVES", etc.							√	

Table 64. COCOMO II SLOC Checklist

Definition Checklist for Source Statements Counts			
Definition name: Logical Source Statements (basic definition)		Date:_____ Originator: COCOMO II	
FORTRAN			
1 END statements		√	
2 Format statements		√	
3 Entry statements		√	
Pascal			
1 Executable statements not terminated by semicolons		√	
2 Keywords like INTERFACE and IMPLEMENTATION		√	
3 FORWARD declarations		√	

Summary of Statement Types

Executable statements

Executable statements cause runtime actions. They may be simple statements such as assignments, goto's, procedure calls, macro calls, returns, breaks, exits, stops, continues, nulls, no-ops, empty statements, and FORTRAN's END. Or they may be structured or compound statements, such as conditional statements, repetitive statements, and "with" statements. Languages like Ada, C, C++, and Pascal have block statements [begin...end and {...}] that are classified as executable when used where other executable statements would be permitted. C and C++ define expressions as executable statements when they terminate with a semicolon, and C++ has a <declaration> statement that is executable.

Declarations

Declarations are nonexecutable program elements that affect an assembler's or compiler's interpretation of other program elements They are used to name, define, and initialize; to specify internal and external interfaces; to assign ranges for bounds checking; and to identify and bound modules and sections of code. Examples include declarations of names, numbers, constants, objects, types, subtypes, programs, subprograms, tasks, exceptions, packages, generics, macros, and deferred constants. Declarations also include renaming declarations, use clauses, and declarations that instantiate generics. Mandatory begin...end and {...} symbols that delimit bodies of programs and subprograms are integral parts of program and subprogram declarations. Language superstructure elements that establish boundaries for different sections of source code are also declarations. Examples include terms such as PROCEDURE DIVISION, DATA DIVISION, DECLARATIVES, END DECLARATIVES, INTERFACE, IMPLEMENTATION, SYS-PROC and SYS-DD. Declarations, in general, are never required by language specifications to initiate runtime actions, although some languages permit compilers to implement them that way.

Compiler Directives

Compiler directives instruct compilers, preprocessors, or translators (but not runtime systems) to perform special actions. Some, such as Ada's pragma and COBOL's COPY, REPLACE, and USE, are integral parts of the source language. In other languages like C and C++, special symbols like # are used along with standardized keywords to direct preprocessor or compiler actions. Still other languages rely on nonstandardized methods supplied by compiler vendors. In these languages, directives are often designated by special symbols such as #, $, and {$}.

Acronyms and Abbreviations

3GL	Third Generation Language
4GL	Fourth Generation Language
A	Effort coefficient that can be calibrated
ATPROD	Automatic translation productivity
AA	Percentage of reuse effort due to assessment and assimilation
AAF	Adaptation Adjustment Factor, a component of the overall Adaptation Adjustment Multiplier for reuse sizing, including the effects of Design Modified, Code Modified, and Integration Modified factors (COCOMO Reuse model).
AAM	Adaptation Adjustment Multiplier for reuse sizing (COCOMO Reuse model)
ACAP	Analyst Capability Cost Driver
APEX	Applications Experience Cost Driver
API	Application Program Interface
ASLOC	Adapted Source Lines of Code, used in reuse sizing (COCOMO Reuse model)
AT	Automated Translation
B	The scaling base-exponent for the effort equation that can be calibrated
C	Coefficient that can be calibrated
CASE	Computer Aided Software Engineering
CCB	Change Control Board
CD	Commercial technology and DoD general practice
CDR	Critical Design Review milestone (Waterfall development process)
CII	COCOMO II.2000
CM	Percentage of code modified during reuse (COCOMO Reuse model)
CM	Configuration Management
CMM	Capability Maturity Model
COCOMO	Constructive Cost Model; refers collectively to COCOMO 81 and COCOMO II.
COCOMO 81	The original version of the Constructive Cost Model, published in 1981
COCOMO II	The revised version of the Constructive Cost Model, first released in 1997
COCOMO II.1997	The original year 1997 calibration of the revised Constructive Cost Model
COCOMO II.2000	The year 2000 calibration of the revised Constructive Cost Model
COCOTS	Constructive COTS cost model
COPROMO	Constructive Productivity improvement Model
COPSEMO	Constructive Phased Schedule & Effort Model
COQUALMO	Constructive Quality Model
CORADMO	Constructive RAD cost model
Cost Driver	A particular characteristic of the software development that has a multiplicative effect of increasing or decreasing the amount of development effort, e.g. required product reliability, execution time constraints, project team application experience.
COTS	Commercial-off-the-shelf

CPLX	Product Complexity Cost Driver
D	The scaling base-exponent for the schedule equation that can be calibrated
DATA	Database Size Cost Driver
DBMS	Database Management System
DM	Percentage of design modified during reuse (COCOMO Reuse model)
DOCU	Documentation Match to Lifecycle Needs Cost Driver
E	The scaling exponent for the schedule equation that can be calibrated
EAF	Effort Adjustment Factor – product of Cost Drivers
EM	Effort Multiplier; a value associated with a specific Cost Driver rating
ESLOC	Equivalent Source Lines of Code for reuse software (COCOMO Reuse model)
F	Scaling exponent for the schedule equation
FCIL	Facilities
FLEX	Development Flexibility scale factor
FP	Function Points
FSP	Full-time Software Personnel
GUI	Graphical User Interface
H	High driver rating
IFPUG	International Function Point Users Group
IM	Integration Modified: percentage of integration and test redone during reuse (COCOMO Reuse model)
IOC	Initial Operational Capability milestone (MBASE/RUP development process)
IECT	Inception, Elaboration, Construction, and Transition phases for the MBASE/RUP lifecycle model
IRR	Initial Readiness Review milestone (MBASE/RUP development process)
KASLOC	Thousands of Adapted Source Lines of Code (COCOMO Reuse model)
KESLOC	Thousands of Equivalent Source Lines of Code (COCOMO Reuse model)
KNCSS	Thousands of Non-Commented Source Statements
KSLOC	Thousands (K) of Source Lines of Code
L	Low driver rating
LCA	Life cycle Architecture milestone (MBASE/RUP development process)
LCO	Life cycle Objectives milestone (MBASE/RUP development process)
LCR	Life cycle Concept Review milestone (MBASE/RUP development process)
LEXP	Programming Language Experience, used in COCOMO 81
LOC	Lines of Code
LTEX	Language and Tool Experience Cost Driver
MAF	Maintenance Adjustment Factor; used to account for software understanding and unfamiliarity effects (COCOMO Reuse and Maintenance models)
MBASE	Model-Based (System) Architecting and Software Engineering
MCF	Maintenance Change Factor: fraction of legacy code modified or added (COCOMO Maintenance model)
Mo	Months
N	Nominal driver rating

NIST	National Institute of Standards and Technology
PCAP	Programmer Capability Cost Driver
PCON	Personnel continuity Cost Driver
PDIF	Platform Difficulty: composite Cost Driver for Early Design model
PDR	Product Design Review milestone (Waterfall development process)
PERS	Personnel Capability: composite Cost Driver for Early Design model
PLEX	Platform Experience Cost Driver
PM	Person-Months; a person month is the amount of time one person spends working on the software development project for one month; in COCOMO normally assumed to be 152 person-hours.
PM_{AUTO}	Person-months effort from automatic translation activities
PM_{NS}	Person-months estimated without the SCED cost driver (Nominal Schedule)
PMAT	Process Maturity scale factor
PR	Productivity Range
PREC	Precedentedness scale factor
PRED(X)	Prediction Accuracy: percentage of estimates within X% of the actuals
PREX	Personnel Experience: composite Cost Driver for Early Design model
PROD	Productivity rate
PVOL	Platform Volatility Cost Driver
RAD	Rapid Application Development; applies to both schedule and effort
RCPX	Product Reliability and Complexity: composite Cost Driver for Early Design model
RELY	Required Software Reliability Cost Driver
RESL	Architecture and Risk Resolution scale factor
REVL	Requirements Evolution and Volatility: size adjustment factor
ROI	Return on Investment
RUP	Rational Software Corporation's Unified Process
RUSE	Developed for Reusability Cost Driver
RVOL	Requirements Volatility, used in COCOMO 81
SAR	Software Acceptance Review milestone (Waterfall development process)
Scale Factor	A particular characteristic of the software development that has an exponential effect of increasing or decreasing the amount of development effort, e.g. precedentedness, process maturity.
SCED	Required Development Schedule: project-level Cost Driver
SCED%	Required Schedule Compression percentage
SECU	Classified Security Application, used in Ada COCOMO
SEI	Software Engineering Institute
SF	Scale Factor; a value for a specific rating of a Scale Factor
SITE	Multi-site Development Cost Driver
SLOC	Source Lines of Code
SRR	Software Requirements Review milestone (Waterfall development process)
STOR	Main Storage Constraint Cost Driver
SU	Percentage of reuse effort due to software understanding (COCOMO Reuse model)

SW-CMM	Software Capability Maturity Model
T&E	Test and Evaluation
TCR	Transition Completion Review milestone (MBASE/RUP development process)
TDEV	Time to Develop (in months)
TEAM	Team Cohesion scale factor
TIME	Execution Time Constraint Cost Driver
TOOL	Use of Software Tools Cost Driver
UNFM	Programmer Unfamiliarity; factor used in reuse and maintenance estimation (COCOMO Reuse and Maintenance models)
USAF/ESD	U.S. Air Force Electronic Systems Division
UTC	Unit Test Completion milestone (Waterfall development process)
VH	Very High driver rating
VL	Very Low driver rating
XH	Extra High driver rating

References

[Banker et al. 1994]. Banker R. D., Chang H., Kemerer C., "Evidence on Economies of Scale in Software Development", Information and Software Technology, 1994, pp. 275-282.

[Behrens 1983]. C. Behrens, "Measuring the Productivity of Computer Systems Development Activities with Function Points," IEEE Transactions on Software Engineering, November 1983.

[Boehm 1981]. B. Boehm, *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, N.J., 1981.

[Boehm-Royce 1989]. B. Boehm, and W. Royce, "Ada COCOMO and the Ada Process Model," Proceedings, Fifth COCOMO Users' Group Meeting, Software Engineering Institute, Pittsburgh, PA, November 1989.

[Boehm 1996]. B. Boehm, "Anchoring the Software Process," IEEE Software, July 1996.

[Boehm et al. 1999]. B. Boehm, D. Port., A. Egyed, and M. Abi-Antoun, "The MBASE Life Cycle Architecture Package: No Architecture is an Island," in P. Donohoe (ed.), *Software Architecture*, Kluwer, 1999, pp. 511-528.

[Boehm-Port 1999]. B. Boehm and D. Port, "Escaping the Software Tar Pit: Model Clashes and How to Avoid Them," *ACM Software Engineering Notes*, Jan. 1999, pp. 36-48.

[Boehm et al. 2000]. Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald Reifer, and Bert Steece, *Software Cost Estimation with COCOMO II*, Prentice Hall, Englewood Cliffs, N.J., to appear June 2000.

[Gerlich-Denskat 1994]. R. Gerlich, and U. Denskat, "A Cost Estimation Model for Maintenance and High Reuse," Proceedings, ESCOM 1994, Ivrea, Italy, 1994.

[Goethert et al.1992]. W. Goethert, E. Bailey, M. Busby, "Software Effort and Schedule Measurement: A Framework for Counting Staff Hours and Reporting Schedule Information." CMU/SEI-92-TR-21, Software Engineering Institute, Pittsburgh, PA.

[IFPUG 1994]. *Function Point Counting Practices: Manual Release 4.0*, International Function Point Users' Group, Blendonview Office Park, 5008-28 Pine Creek Drive, Westerville, OH 43081-4899.

[Jacobson et al. 1999]. I. Jacobson, G. Booch and J. Rumbaugh, *The Unified Software Development Process*, Addison Wesley Longman, Reading, Ma., 1999.

[Jones 1996]. C. Jones, *Applied Software Measurement, Assuring Productivity and Quality*, McGraw-Hill, New York, N.Y, 1996.

[Kruchten 1999]. P. Kruchten, *The Rational Unified Process: An Introduction*, Addison-Wesley, 1999.

[Kunkler 1983]. J. Kunkler, "A Cooperative Industry Study on Software Development / Maintenance Productivity," Xerox Corporation, Xerox Square --- XRX2 52A, Rochester, NY 14644, Third Report, March 1985.

[Marenzano 1995]. J. Marenzano, "System Architecture Validation Review Findings," in D. Garlan, ed., ICSE17 Architecture Workshop Proceedings, CMU, Pittsburgh, PA 1995.

[Park 1992]. R. Park, "Software Size Measurement: A Framework for Counting Source Statements." CMU/SEI-92-TR-20, Software Engineering Institute, Pittsburgh, PA, 1992.

[Paulk et al. 1995]. M. Paulk, C. Weber, B. Curtis, and M. Chrissis, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, 1995.

[Parikh-Zvegintzov 1983]. G. Parikh, and N. Zvegintzov, "The World of Software Maintenance," Tutorial on Software Maintenance, IEEE Computer Society Press, pp. 1-3, 1995.

[Royce 1998]. R. Royce, *Software Project Management A Unified Framework*, Addison-Wesley, Reading, Ma., 1998.

[Ruhl-Gunn 1991]. M. Ruhl and M. Gunn, "Software Reengineering: A Case Study and Lessons Learned," NIST Special Publication 500-193, Washington, DC, September 1991.

[Selby 1988]. R. Selby, "Empirically Analyzing Software Reuse in a Production Environment," In Software Reuse: Emerging Technology, W. Tracz (Ed.), IEEE Computer Society Press, 1988., pp. 176-189.

[Stensrud 1998]. E. Stensrud, "Estimating with Enhanced Object Points vs. Function Points," *Proceedings, 13th COCOMO/SCM Forum*, USC, October 1998.