The Rosetta Stone: Making COCOMO 81 Files Work With COCOMO II

Donald J. Reifer, Reifer Consultants, Inc.
Barry W. Boehm, University of Southern California
Sunita Chulani, University of Southern California

As part of our efforts to help COCOMO users, we, the COCOMO research team at the Center for Software Engineering at the University of Southern California (USC), have developed the Rosetta Stone for converting COCOMO 81 files to run using the new COCOMO II software cost estimating model. The Rosetta Stone is very important because it allows users to update estimates made with the earlier version of the model so that they can take full advantage of the many new features incorporated into the COCOMO II package. This paper describes both the Rosetta Stone and guidelines for making the job of conversion easy.

Setting the Stage

During the past few years, the COCOMO team at the University of Southern California (USC) has been working to update the 1981 version of the COnstructive COst MOdel (COCOMO) estimating model (COCOMO 81)[1]. The new version of the model called COCOMO II builds on the experiences that industrial affiliates of the USC Center for Software Engineering have had and addresses life cycle processes and paradigms that have become popular since the original model was first introduced in 1981. These new paradigms include reuse-driven approaches, commercial off-the-shelf (COTS) life cycle developments, component-based software engineering approaches, use of object-oriented methods and a variety of other improvements to the way we do business stimulated by process maturity initiatives.

In this article, we will focus attention on a tool we have developed to permit our users to update their original COCOMO 81 files so that they can be used with the COCOMO II model. We call the tool the Rosetta Stone because it is not unlike the black slab found by French troops in 1799 in Egypt containing three scripts (Greek, demotic and hieroglyphics) that enabled archaeologists to construct translations among the three languages. This Rosetta Stone permits its users to translate files prepared using the original COCOMO 81 model to a form compatible with COCOMO II.

You are probably asking why our affiliates thought creation of the Rosetta Stone was important. Many of them wanted to use the new version of the model to take advantage of its many advanced capabilities including the COCOMO II package's auto-calibration features. But, they couldn't make the move because they had files that required older versions of model to run (original COCOMO, Ada-COCOMO, etc.). Others wanted to calibrate the new version of the model using their historical databases. But, the new version of the model had a new structure, altered mathematics and different parameters and parametric ratings. Under such circumstances, converting files was no easy task.

The COCOMO II Estimating Model

Based upon our introduction, you are probably asking "What are the major differences between COCOMO 81 and COCOMO II and why are they important?" Model differences are summarized in Table 1. These changes are important because they reflect how the state of software engineering technology has matured during the past two decades. For example, programmers were submitting batch jobs when the COCOMO 81 model was first published. Turnaround time impacted their productivity. Therefore, a parameter TURN was used in the model to reflect the average wait a programmer experienced prior to receiving their job back. Such a parameter is no longer important because most programmers have instant access to computational facilities through their workstation. Therefore, the parameter has been removed in the COCOMO II model.

The following summary highlights the major changes made to the original version of COCOMO 81 as COCOMO II was developed:

- The COCOMO II addresses the following three phases of the spiral life cycle: applications development, early design and post-architecture.

- The three modes in the exponent are replaced by five Scale Factors.

- The following cost drivers were added to COCOMO II: DOCU, RUSE, PVOL, PEXP, LTEX, PCON and SITE.

- The following cost drivers were deleted from the original COCOMO: VIRT, TURN, VEXP, LEXP and MODP.

- The ratings for those cost drivers retained in COCOMO II were altered considerably to reflect more up-to-date calibrations.

The Rosetta Stone

As illustrated in Table 2, users need to convert factors in the COCOMO equations (i.e., the exponent, the size estimate and the ratings for the cost drivers) from the original to the new version of the model. We suggest that users employ the following four steps to make the conversion so original files can be used with the COCOMO II model:

- **Update size** - The original COCOMO model used deliverable source lines of code (DSI) as its measure of the size of the software job. DSI were represented by card images (e.g., includes all non-blank, non-comment carriage returns). COCOMO II uses the following three different measures to bound the volume of work associated with a software job: source lines of code (SLOC's), function points and object points. SLOC's are counted using logical language statements per Software Engineering Institute guidelines[2] (e.g., IF-THEN-ELSE, ELSE IF is considered a single not two statements). Table 2 provides guidelines for converting size in DSI to SLOC's for their use with the COCOMO II model. Whenever possible, we suggest that you use counts for the actual size for the file instead of the original estimate. Such practices allow you to correlate your actuals (e.g., the actual application size with the effort required to do the work associated with developing the software).

Table 1 Model Comparisons

	COCOMO 81	COCOMO II
Model structure	Single model which assumes you start with requirements allocated to software	Three models which assume you progress through a spiral type development to solidify your requirements, solidify the architecture and reduce risk
Mathematical form of effort equation	$\text{Effort} = A(c_i)\,(\text{Size})^{\text{Exponent}}$	$\text{Effort} = A(c_i)\,(\text{Size})^{\text{Exponent}}$
Exponent	Exponent = fixed constant selected as a function of mode - Organic = 1.05 - Semi-detached = 1.12 - Embedded = 1.20	Exponent = variable established based upon rating of five scale factors - **PREC**, Precedentedness - **FLEX**, Development Flexibility - **RESL**, Architecture/Risk Resolution - **TEAM**, Team Cohesion - **PMAT**, Process Maturity
Size	Source lines of code (with extensions for function points)	Object points, function points or source lines of code
Cost drivers (c_i)	Fifteen drivers each of which must be rated: - **RELY**, Reliability - **DATA**, Data Base Size - **CPLX**, Complexity - **TIME**, Execution Time Constraint - **STOR**, Main Storage Constraint - **VIRT**, Virtual Machine Volatility - **TURN**, Turnaround Time - **ACAP**, Analyst Capability - **PCAP**, Programmer Capability - **AEXP**, Applications Experience - **VEXP**, Virt. Machine Experience - **LEXP**, Language Experience - **TOOL**, Use of Software Tools - **MODP**, Use of Modern Programming Techniques - **SCED**, Required Schedule	Seventeen drivers each of which must be rated - **RELY**, Reliability - **DATA**, Data Base Size - **CPLX**, Complexity - **RUSE**, Required Reusability - **DOCU**, Documentation - **TIME**, Execution Time Constraint - **STOR**, Main Storage Constraint - **PVOL**, Platform Volatility - **ACAP**, Analyst Capability - **PCAP**, Programmer Capability - **AEXP**, Applications Experience - **PEXP**, Platform Experience - **LTEX**, Language & Tool Experience - **PCON**, Personnel Continuity - **TOOL**, Use of Software Tools - **SITE**, Multi-site Development - **SCED**, Required Schedule
Other model differences	Model based upon: - Linear reuse formula - Assumption of reasonably stable requirements	Has many other enhancements including: - Non-linear reuse formula - Reuse model which looks at effort needed to understand and assimilate - Breakage ratings which are used to address requirements volatility - Auto-calibration features

Table 2 Converting Size Estimates

COCOMO 81	COCOMO II
DSI - 2nd generation languages - 3rd generation languages - 4th generation languages - object-oriented languages	SLOC[3] - reduce DSI by 35% - reduce DSI by 25% - reduce DSI by 40% - reduce DSI by 30%
Function points	Use the expansion factors developed by Capers Jones[4] to determine equivalent SLOC's
Feature points	Use the expansion factors developed by Capers Jones to determine equivalent SLOC's

The size reduction in COCOMO II is attributable to need to convert card images to source line of code counts. As already noted, the pair IF-THEN-ELSE and END IF would be counted as either two card images in COCOMO 81 and as a single source instruction in COCOMO II. The guidelines offered in Table 2 are based on statistical averages in order to simplify conversions. However, we encourage you to use your actuals if you have them at your disposal.

We would like to address the following two misconceptions about COCOMO use of source lines of code (SLOC) and function points (FP):

- **Misconception 1: COCOMO does not support the use of function points**- Function point versions of COCOMO have been available since the *Before You Leap* commercial COCOMO software package implementation in 1987. As noted in Table 1, COCOMO II supports use of either SLOC or FP metric. In both cases, this is done via "backfiring" tables which permit you to convert function points to lines of code at different levels.

- **Misconception 2: It is irresponsible to use SLOC as a general productivity metric, but it is not irresponsible to use FP as a general sizing parameter for estimation** - This misconception breaks down into the two following cases:

  - Your organization uses different language levels to develop software. In this case, it is irresponsible to use SLOC as your productivity metric, as you get higher productivity/SLOC at higher language levels. However, it is also irresponsible to use FP as a general sizing metric because pure FP will generate the same cost (or schedule or quality) estimate for a program with the same functionality developed using different language levels. This is clearly wrong. To get responsible results in this case, FP-based estimation models need to use some form of backfiring to account for the difference in language level.

  - Your organization always uses the same programming language (level). Here, it is responsible to use pure FP as your sizing metric for estimation. But, it is also responsible to use SLOC as your productivity metric. Both metrics work in practice.

- **Convert exponent** - Convert the original COCOMO 81 modes to Scale Factor settings using the Rosetta Stone values in Table 3. Then, adjust the ratings to reflect the actual situation. For example, the Rosetta Stone rates PMAT low because most projects using COCOMO 81 are assumed to have been at level 1 on the Software Engineering Institute (SEI) process maturity scale[5]. However, the project's actual rating may have been higher and an adjustment may be in order.

Table 3 Mode/Scale Factor Conversion Ratings

MODE/SCALE FACTORS	ORGANIC	SEMI-DETACHED	EMBEDDED
Precedentedness (PREC)	XH	H	L
Development flexibility (FLEX)	XH	H	L
Architecture/risk resolution (RESL)	XH	H	L
Team cohesion (TEAM)	XH	VH	N
Process maturity (PMAT)	MODP	MODP	MODP

An exception is the process Maturity (PMAT) scale factor, which replaces the COCOMO 81 Modern Programming Practices (MODP) cost driver. As seen in Table 4, MODP ratings of VL or L translate into a PMAT rating of VL, or a low level on the SEI CMM scale. A MODP rating of N translates into a PMAT rating of L, or a high Level 1. A MODP rating of H or VH translates into a PMAT rating of N or CMM Level 2. As with the other factors, if you know that the project's actual rating was different from the one provided by the Rosetta Stone, use the actual value.

The movement from modes to scale factors represents a major change in the model. To determine the economies/diseconomies of scale, five factors have been introduced. Because each of these factors can influence the power to which size is raised in the COCOMO equation, they can have a profound impact on cost and productivity. For example, increasing the rating from H to VH in these parameters can introduce as much as a six percent swing in the resulting resource estimate. Most of these factors are modern in their derivation. For example, the concept of process maturity wasn't even in its formative stages when the original COCOMO 81 model was published. In addition, the final three factors RESL, TEAM and PMAT show how an organization can exercise management control over its diseconomies of scale. Finally, the first two, PREC and FLEX, are the less controllable factors contributing to COCOMO 81 mode definitions.

- **Rate Cost Drivers** - The trickiest part of the conversion is the cost drivers. Cost drivers are parameters to which cost is sensitive. For example, as with the scale factors you would expect use of experienced staff would make a software development less expensive. Else, why use them? Because the new version of the model uses altered drivers, the Rosetta Stone conversion guidelines outlined in Table 4 are important. For those interested in more details about the cost drivers, we suggest that you refer to the COCOMO II Model Definition Manual[6]. Again, the ratings need to be adjusted to reflect what actually happened on the project. For example, the original estimate may have assumed that applications experience was very high. However, the caliber of analysts actually assigned might have been nominal because key personnel were not available to the project when they were needed.

Table 4 Cost Drivers Conversions

COCOMO 81 DRIVERS	COCOMO II DRIVERS	CONVERSION FACTORS
RELY	RELY	None, rate the same or the actual
DATA	DATA	None, rate the same or the actual
CPLX	CPLX	None, rate the same or the actual
TIME	TIME	None, rate the same or the actual
STOR	STOR	None, rate the same or the actual
VIRT	PVOL	None, rate the same or the actual
TURN		Use values in Table 5
ACAP	ACAP	None, rate the same or the actual
PCAP	PCAP	None, rate the same or the actual
VEXP	PEXP	None, rate the same or the actual
AEXP	AEXP	None, rate the same or the actual
LEXP	LTEX	None, rate the same or the actual
TOOL	TOOL	Use values in Table 5
MODP	Adjust PMAT settings	If MODP is rated VL or L, set PMAT to VL N, set PMAT to L H or VH, set PMAT to N
SCED	SCED	None, rate the same or the actual
	RUSE	Set to N, or actual if available
	DOCU	If Mode = Organic, set to L = Semi- Detached, set to N = Embedded, set to H
	PCON	Set to N, or actual if available
	SITE	Set to H, or actual if available

Users should take advantage of their actual knowledge of what occurred on the project to make their estimates more reflective of what really went on as the application was developed. Use of such knowledge can improve the credibility and accuracy of their estimates.

The TURN and TOOL rating scales have been affected by technology changes since 1981. These days, virtually everyone uses interactive workstations to develop software. TURN has therefore been dropped from COCOMO II and its calibration assumes the TURN rating is L. Table 5 provides alternative multipliers for other COCOMO 81 TURN ratings.

The tool suites available in the 1990's far exceed the COCOMO 81 VH TOOL rating and virtually no projects operate at the COCOMO 81 VL or L TOOL levels. COCOMO II has shifted the TOOL rating scale two levels higher so that a COCOMO 81 N TOOL rating corresponds to a VL COCOMO II TOOL rating. Figure 5 also provides a set of COCOMO II multipliers corresponding to COCOMO 81 project ratings.

Some implementations of COCOMO II, such as the USC COCOMO II package, provide slots for extra user defined cost drivers. The values in Figure 5 can be put into such slots (if you do this, use a N rating in the normal COCOMO II TOOL slot).

Table 5   TURN and TOOL Adjustments

COCOMO II MULTIPLIER/COCOMO 81 RATING	VL	L	N	H	VH
TURN		1.00	1.15	1.23	1.32
TOOL			1.24	1.10	1.00

For those interested in learning more about the cost drivers and their ratings, we refer you to the USC web site (the URL is http://sunset.usc.edu/COCOMOII) or several of the Center for Software Engineering's other publications[7,8]. Because the goal of this article is to present the Rosetta Stone, we did not feel it was really necessary to go into the details of the model and an explanation of its many parameters.

Experimental Accuracy

To assess the accuracy of the translations, the team used the Rosetta Stone to convert 89 projects. These projects were clustered subsets of the databases we used for model calibration. Clusters were domain specific. We updated our estimates using actuals whenever we could. We then used the auto-calibration feature of the USC COCOMO II package to develop a constant for the effort equation (i.e., the **A** in the equation: Effort = $\underline{\mathbf{A}}(\mathrm{SIZE})^{P}$). Finally, we compared our estimates to actuals and computed the relative error as a function of the following cases:

- Using the Rosetta Stone with no adjustments,
- Using the Rosetta Stone with knowledge base adjustments (i.e., updating the estimate files with actuals when available), and
- Using the Rosetta Stone with knowledge base adjustments and domain clustering (i.e., segmenting the data based upon organization or application area).

The results of these analyses, which are summarized in Table 6, were very positive. They show that we can achieve an acceptable degree of estimating accuracy when using the Rosetta Stone to convert COCOMO 81 files to run with the COCOMO II software cost model.

Table 6   Estimate Accuracy Analysis Results

CASES	ACCURACY (RELATIVE ERROR)
Using the COCOMO II model as calibrated	Estimates within 25% of actuals, 68% of the time
Using the COCOMO II model as calibrated using developer or domain clustering	Estimates within 25% of actuals, 76% of the time
Using Rosetta Stone with no adjustments	Estimates within 25% of actuals, 60% of the time
Using the Rosetta Stone with knowledge base adjustments	Estimates within 25% of actuals, 68% of the time
Using the Rosetta Stone with knowledge base adjustments and domain clustering	Estimates within 25% of actuals, 74% of the time

Summary and Conclusions

The Rosetta Stone was developed to provide its users with both a process and tool for converting their original COCOMO 81 files so that they can be used with the new COCOMO II estimating model. The Stone represents a starting point for such efforts. It does not replace the need to understand either the scope of the estimate or the changes that occurred as the project unfolded. Rather, the Stone takes these considerations into account as you update its knowledge base with actuals.

The value of the Rosetta Stone was demonstrated convincingly based upon an accuracy analysis of an eighty-nine project database. As expected, the accuracy increased as we adjusted the estimates using actuals and looked at results based upon domain segmentations. We are encouraged by the results. We plan to continue our efforts to provide structure and support for such conversion efforts.

References

1. Boehm, B. (1981), *Software Engineering Economics,* Prentice-Hall, Englewood Cliffs, NJ.
2. Park, R. (1992), "Software Size Measurement: A Framework for Counting Source Statements, CMU/SEI-92-TR-20, Software Engineering Institute, Pittsburgh, PA.
3. Reifer, D. (1998), personal correspondence.
4. Jones, C. (1991), *Applied Software Measurement*, *Assuring Productivity and Quality*, McGraw-Hill, New York, NY.
5. Paulk, M., C. Weber, B. Curtis, M. Chrissis (1995*)*, *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison-Wesley, Reading, MA.
6. Boehm, B., et. al., (1997), *COCOMO II Model Definition Manual, Version 1.4,* University of Southern California, Los Angeles, CA.
7. Boehm, B., et. al. (1996), "The COCOMO 2.0 Software Cost Estimation Model," *American Programmer*, pp. 2-17.
8. Clark, B. and D. Reifer (1997), "The Rosetta Stone: Making Your COCOMO Estimates Work with COCOMO II," *Software Technology Conference*, Salt Lake City, UT, 1998.

About the Authors

Donald J. Reifer is one of the leading figures in the fields of software engineering and management, with over 30 years of progressive experience in both government and industry. From 1993 to 1995, he was Chief of the Ada Joint Program Office, technical advisor to the Center for Software, and Director of the DoD Software Reuse Initiative under an Intergovernmental Personnel Act assignment with the Defense Information Systems Agency. Currently, Reifer serves as President of RCI, a small firm that specializes in helping Fortune 500 clients improve the way they do business. Mr. Reifer is a visiting associate at the University of Southern California where he serves on the COCOMO team.

Reifer holds a BS degree in Electrical Engineering, an MS degree in Operations Research, and a Certificate in Business Management (MBA equivalent). His many honors include the Secretary of Defense's medal for Outstanding Public Service, the NASA Distinguished Service Medal, the Freiman award, and the Hughes Aircraft Fellowship. Reifer has over 100 publications including his popular ***IEEE Software Management Tutorial (5th Edition)*** and his new Wiley book entitled ***Practical Software Reuse***.

Donald J. Reifer, President
Reifer Consultants, Inc.
P.O. Box 4046,
Torrance, CA 90505
Phone: 310-530-4493
E-mail: d.reifer@ieee.org

Dr. Barry Boehm is considered one of the fathers of the field of software engineering. Currently, he serves as the Director of the Center for Software Engineering at the University of Southern California. Between 1989 and 1992, he served within the U.S. Department of Defense (DOD) as Director of the DARPA Information Science and Technology Office, and as Director of the DDR&E Software and Computer Technology Office. He worked at TRW from 1973 to 1989 as the Chief Scientist of the Defense Systems Group and at the Rand Corporation from 1959 to 1973 as Head of the Information Sciences Department. He was a Programmer-Analyst at General Dynamics between 1955 and 1959. His current research interests include software process modeling, software requirements engineering, software architectures, software metrics and cost models, software engineering environments, and knowledge-based software engineering. His contributions to the field include the Constructive Cost Model (COCOMO), the Spiral Model of the software process, the Theory W approach to software management and requirements determination, and the TRW Software Productivity System and Quantum Leap advanced software engineering environments.

Boehm received his BA degree from Harvard in 1957, and his MS and Ph.D. degrees from UCLA in 1961 and 1964, all in Mathematics. He has served on the board of several scientific journals, including the IEEE Transactions on Software Engineering, IEEE Computer, IEEE Software and ACM Computing Reviews. He has served as Chair of the AIAA Technical Committee on Computer Systems, Chair of the IEEE Technical Committee on Software Engineering, and as a member of the Governing Board of the IEEE Computer Society. He currently serves as Chair of the Board of Visitors for the Software Engineering Institute. His many honors and awards include the AIAA Information Systems Award (1979), the J.D. Warnier Prize for Excellence in Information Sciences (1984), the ISPA Freiman Award for Parametric Analysis (1988), the NSIA Grace Murray Hopper Award (1989), the Office of the Secretary of Defense Award for Excellence (1992), the ASQC Lifetime Achievement Award (1994), and the ACM Distinguished Research Award in Software Engineering (1997). He is an AIAA Fellow, an ACM Fellow, an IEEE Fellow, and a member of the National Academy of Engineering.

Dr. Barry W. Boehm, Director
Center for Software Engineering
University of Southern California
941 West 37th Place
Los Angeles, CA 90089
Phone: 213-740-8163
E-mail: boehm@sunset.usc.edu

Ms. Sunita Chulani is a research assistant at the Center for Software Engineering at the University of Southern California. She is an active participant of the COCOMO II research team and is working on a Bayesian approach to data analysis and model calibration. She is also working on a Cost/Quality Model that will be an extension to the existing COCOMO II model. Her main interests include software process improvement with statistical process control, software reliability modeling, risk assessment, software cost estimation and software metrics.

Ms. Chulani holds a BE in Computer Engineering from Bombay University and an MS in Computer Science from the University of Southern California. She is currently a Ph.D. candidate at the Center for Software Engineering at the University of Southern California.

Ms. Sunita Chulani
Center for Software Engineering
University of Southern California
941 West 37th Place
Los Angeles, CA 90089
Phone: 213-740-6470
E-mail: sdevnani@sunset.usc.edu