COCOTS: A COTS Software Integration Lifecycle Cost Model - Model Overview and Preliminary Data Collection Findings

Chris Abts, M.S.
University of Southern California
Salvatori Hall Room 328
941 W. 37th Place
Los Angeles, CA 90089 USA
+1 213 740 6470
cabts@sunset.usc.edu

Barry W. Boehm, Ph.D.
University of Southern California
Salvatori Hall Room 328
941 W. 37th Place
Los Angeles, CA 90089 USA
+1 213 740 5703
boehm@sunset.usc.edu

Elizabeth Bailey Clark, Ph.D.
Software Metrics, Inc.
4345 High Ridge Road
Haymarket, VA 20169 USA
+1 703 754 0115
BetsyClark@erols.com

ABSTRACT
As the use of commercial-of-the-shelf (COTS) components becomes ever more prevalent in the creation of large software systems, the need for the ability to reasonably predict the true lifetime cost of using such software components grows accordingly. In using COTS components, immediate short-term gains in direct development effort & schedule are possible, but usually as a trade-off for a more complicated long-term post-deployment maintenance environment. In addition, there are risks associated with COTS software separate from those of creating components from scratch. These unique risks can further complicate the development and post-deployment situations. This paper discusses a model being developed as an extension of the COCOMO II [1] cost model. COCOTS attempts to predict the lifecycle costs of using COTS components by capturing the more significant COTS risks in its modeling parameters. The current state of the model is presented, along with some preliminary findings suggested by an analysis of calibration data collected to date. The paper concludes with a discussion of the on-going effort to further refine the accuracy and scope of COCOTS.

Keywords
COTS, COTS integration, COTS assessment, COTS software lifecycle, COCOMO II, cost estimation, effort and schedule estimation, metrics, software engineering

1 INTRODUCTION

COCOTS is the acronym for the *COnstructive COTS* integration cost model, where COTS in turn is short for *commercial-off-the-shelf*, and refers to those pre-built, commercially available software components that are becoming ever more important in the creation of new software systems.

The rationale for building COTS-containing systems is that they will involve less development time by taking advantage of existing, market proven, vendor supported products, thereby reducing overall system development costs. But there are two defining characteristics of COTS software, and they drive the whole COTS usage process:

1) the COTS product source code is not available to the application developer, and

2) the future evolution of the COTS product is not under the control of the application developer.

(Note: in some cases, COTS software does come as source code, but for the purposes of our model, if the developer does anything to that code other than compile it unchanged into his own code, we treat that as a *reuse* item and model its usage as *adapted* code [2] within COCOMO II itself.)

Because of these characteristics, there is a trade-off in using the COTS approach in that new software development time can indeed be reduced, but generally at the cost of an increase in software component integration work. The long term cost implications of adopting the COTS approach are even more profound, because you are in fact adopting a new way of doing business from the moment you start considering COTS components for your new system to the day you finally retire that system. This is because COTS software is not static, it continually evolves in response to the market, and you as the system developer must adopt methodologies that cost-effectively manage the use of those evolving components.

The fact is that using COTS software brings with it a host of unique risks quite different from those associated with software developed in-house.

Included among those risks or factors which should be

Table 1 – COTS Advantages and Disadvantages [3]

Advantages	Disadvantages
Immediately available; earlier payback	Licensing, intellectual property procurement delays
Avoids expensive development	Up-front licensing fees
Avoids expensive maintenance	Recurring maintenance fees
Predictable, confirmable license fees and performance	Reliability often unknown or inadequate; scale difficult to change
Rich functionality	Too-rich functionality compromises usability, performance
Broadly used, mature technologies	Constraints on functionality, efficiency
Frequent upgrades often anticipate organization's needs	No control over upgrades and maintenance
Dedicated support organization	Dependence on vendor
Hardware/software independence	Integration not always trivial; incompatibilities among vendors
Tracks technology needs	Synchronizing multiple-vendor upgrades

examined when determining the true cost of integrating a COTS software component into a larger system are not only the traditional costs associated with new software development such as the cost of requirements definition, design, code, test, and software maintenance, but also the cost of licensing and redistribution rights, royalties, effort needed to understand the COTS software, pre-integration assessment and evaluation, post-integration certification of compliance with mission critical or safety critical requirements, indemnification against faults or damage caused by vendor supplied components, and costs incurred due to incompatibilities with other needed software and/or hardware.

Because of these unique risks, using COTS components in the development of new systems is not the universal solution to reducing cost and schedule while maintaining desired quality and functionality. However, if these risks can be managed, using COTS components can frequently be the right solution, offering the most cost-effective, shortest schedule approach to assembling major software systems.

Table 1 summarizes some of the more significant factors that represent possible trade-offs that may need to be considered when using COTS components.

2 RELATION TO COCOMO II

The software development cost estimation model COCOMO (*COnstructive COst MOdel*) was originally published in 1981, and has recently been reincarnated as COCOMO II to reflect current software development practice. It creates effort and schedule estimates for software systems built using a variety of techniques or approaches. The first and primary approach modeled by COCOMO is the use of system components that are built from scratch, that is, *new* code. But COCOMO II also allows you to model the case in which system components are built out of pre-existing source code that is modified or adapted to your current purpose, i.e., *reuse* code. The key word in the preceding sentence is *source*. Even though you are not building the reuse component from scratch, you still have access to the component's source code and can rewrite or modify it specifically to suit your needs.

What COCOMO II currently does not model is that case in which you do not have access to a pre-existing component's source code. You have to take the component as is, working only with its executable file, and at most are able to build a software shell *around* the component to adapt its functionality to your needs.

This is where COCOTS comes in. COCOTS is being designed specifically to model many of the unique conditions and practices highlighted in the preceding section that obtain when you incorporate COTS components into the design of your larger system.

(Note: the model being described in this paper is a work in progress. At the moment it is presented as something wholly separate from COCOMO II, but in its mature incarnation it is anticipated that COCOTS will be folded directly into some future release of the available USC COCOMO II tool, providing a unified estimating tool with the capability of modeling all modes of software system development: new code, reuse code, and COTS component usage.)

3 COCOTS MODEL OVERVIEW

Modeling Methodology

In any engineering or scientific discipline, when developing predictive models, access to empirical data upon which to build those models is key. However, regardless of the discipline, it is also true that acquiring such data is usually a difficult, costly, time and effort-consuming task. In the field of software engineering, particularly its sub-discipline of software development cost estimation, data acquisition can be doubly daunting because of the often lack of obvious or at least straightforward techniques to collect needed data. Moreover, data that is collected can be very subjective, a function of the reality that software

development is in many ways still as much art as engineering. Such data gathering difficulties make developing software cost estimation models particularly challenging. To get around this problem, the researchers at the USC Center for Software Engineering have evolved a multi-step modeling methodology [5, 6] that we have found very useful for developing software estimation models when the amount of related empirical data is initially minimal. Using Bayesian statistical techniques, this approach allows us to establish initial model parameter values based on a blending of numbers derived from expert judgment with calibration numbers derived from whatever empirical data is at hand. Illustrated in Figure 1, this method has allowed us to go forward with developing reasonable model definitions until such time data becomes available for model refinement and calibration.

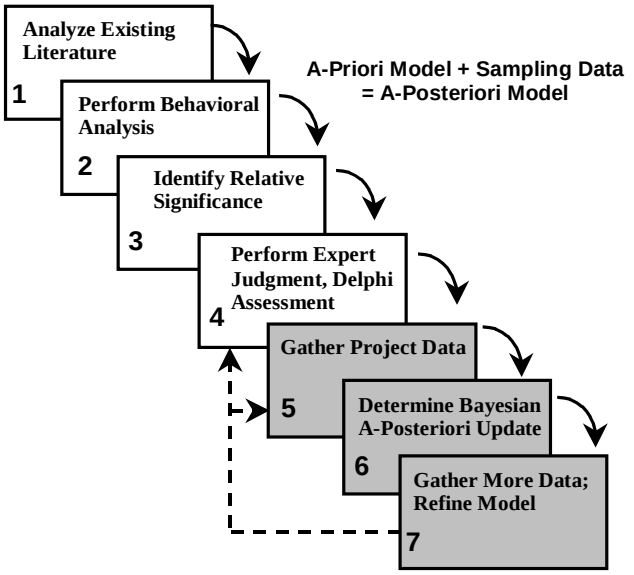

Figure 1. USC-CSE Modeling Methodology

This is the approach we undertook to develop COCOTS. Beginning with step 1, a standard literature search to determine what COTS modeling techniques were already available, we then moved through steps 2 and 3, doing a behavioral analysis to determine the most important activities people actually perform while integrating COTS components. Next, step 4, we gathered a panel of industry experts to help us with establishing initial numerical values for our parameters [7]. We are now iterating on steps 5 through 7, expanding our collection of calibration data while using analyses of that data as it comes in to refine the model and expand its scope.

Model Description

COCOTS at the moment is composed of four related submodels, each addressing individually what we have identified as the four primary sources of COTS software integration costs. (This is a key point. COCOTS as described in this paper deals only with *initial* integration efforts. We have already begun taking the first steps toward expanding the model to cover long-term lifecycle and maintenance costs associated with using COTS components. However, aside from a brief discussion at the end of this paper, that topic will remain largely outside the scope of this paper.)

Initial integration costs are due to the effort needed to perform (1) candidate COTS component assessment, (2) COTS component tailoring, (3) the development and testing of any integration or "glue" code (sometimes called “glueware” or “binding” code) needed to plug a COTS component into a larger system, and (4) increased system level programming and testing due to volatility in incorporated COTS components.

(A fifth cost source was actually identified early in our research. This was the cost related to the increased verification and validation effort unrelated to COTS volatility but still usually required when using COTS components. But attempts have been made to capture these costs within the glue code and tailoring submodels directly rather than specify a fifth independent submodel.)

Assessment is the process by which COTS components are selected for use in the larger system being developed. *Tailoring* refers to those activities that would have to be performed to prepare a particular COTS program for use, regardless of the system into which it is being incorporated, or even if operating as a stand-alone item. These are things such as initializing parameter values, specifying I/O screens or report formats, setting up security protocols, etc. *Glue code* development and testing refers to the new code external to the COTS component itself that must be written in order to plug the component into the larger system. This code by nature is unique to the particular context in which the COTS component is being used, and must not be confused with tailoring activity as defined above. *Volatility* in this context refers to the frequency with which new versions or updates of the COTS software being used in a larger system are released by the vendors over the course of the system's development and subsequent deployment.

It should also be noted that while the assessment submodel lends itself easily to use very early in the project planning stages, the tailoring, glue code, and volatility submodels by the very nature of the costs they are trying to address are more problematic if used before candidate COTS products that may actually be integrated have been identified. The reason is that the costs covered by these models are extremely dependent on the unique characteristics of any given set of COTS products and their vendors.

Scope and Lifecycle Presently Addressed

COCOTS currently addresses only the cost of *software*

COTS components. But the cost associated with hardware COTS elements used in a system may yet be addressed in future refinements of the model, since it is often the case that hardware/software COTS options are bound together; i.e., choosing one dictates the use of another. (In these cases, our data gathering experience suggests that it is still the hardware choice that more often than not leads the software decision rather than the other way around.)

As for lifecycle, as noted previously, as of now COCOTS addresses only initial development costs associated with using COTS components. Incorporating long-term COTS operation and maintenance costs is under examination.

In terms of a *waterfall* development process, the specific project phases currently covered by the model are these:

- Requirements Definition
- Preliminary Code Design
- Detailed Code Design and Unit Test
- Integration and Test

The inclusion of the requirements definition phase at the top of this list is significant. This phase traditionally has NOT been covered by COCOMO estimates. The fact that assessment and requirements definition must be done in tandem when using COTS components if you are to realize the full benefit of adopting the COTS approach to system development necessitates the inclusion of the requirements definition phase in COCOTS estimates.

In terms of a *spiral* development process, COCOTS covers the following lifecycle anchor points [8]:

- Inception
  → Lifecycle Objectives (LCO)
- Elaboration
  → Lifecycle Architecture (LCA)
- Construction
  → Initial Operational Capability (IOC)

Sources of Effort

In terms of level of effort associated with the four primary COTS integration cost sources identified above, the data we've collected to date suggests that the greatest COTS integration effort tends to be concentrated on glue code development. One likely reason for this is that the writing of glue code is constrained in ways the writing of new code is not. Another reason may be that difficulties that must be addressed in the glue code often do not show up until the COTS integration process is well underway, making backtracking of designs and rework a necessity.

The next most significant source of effort as suggested by our data tends to be tailoring. The fact that tailoring as we've defined it is generally a lesser effort than glue code development seems intuitive. Tailoring activities can be involved, but are not usually overly complex. They often follow templates and standard procedures. The writing of glue code on the other hand can require great creativity and ingenuity on the part of the programmer.

Assessment tends to be the least significant source of COTS integration effort, though still measurably important. Again, however, that this would generally comprise the smallest effort overall is intuitive. Projects that get bogged down in the assessment phase are not making much progress towards delivery. (As an aside, however, as we've collected our data, is has become apparent that the most successful projects have *not* given the assessment phase short shrift. They've spent the effort necessary to verify that COTS products will do *exactly* what the vendors say they will do.)

As for the level of effort associated with managing the COTS volatility effects in the larger system, it appears to be larger than that associated with assessment, but its true significance as reflected in our data is still unclear to us, and suggests that this may be one area where our model definition still needs further refinement.

It also needs to be pointed out that the above relative effort orderings are only generalities. The true size of each effort that can be expected during a given software development is highly dependent on the design specifics of the given system. During data collection, we have come across projects in which only tailoring was done and no glue code was written, and vice versa. (No projects, however, got away without expending at least some effort on product assessment.)

Figure 2 on the following page illustrates how the modeling of these effort sources in COCOTS is related to effort modeled by COCOMO II. The figure represents the total effort to build a software system out of a mix of new code and COTS components as estimated by a combination of COCOMO II and COCOTS. The central block in the diagram indicates the COCOMO II estimate, that is, the effort associated with any newly developed software in the system. . The smaller, exterior blocks indicate COCOTS estimates, that effort associated with the COTS components in the system. The relative sizes of the various blocks in this figure is a function of the number of COTS components relative to the amount of new code in the system, and of the nature of the COTS component integration efforts themselves. The more complex the tailoring and/or glue code-writing efforts, the larger these blocks will be relative to the assessment block. Also, note that addressing the system wide volatility due to volatility in the COTS components is an effort that will obtain throughout the entire system development cycle, as indicated by the long block running along the bottom of the figure.

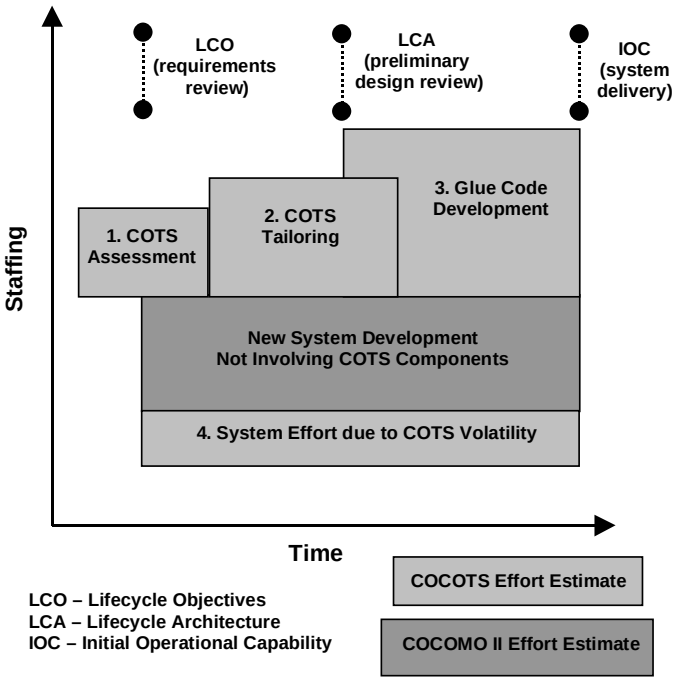

Figure 2. Sources of Effort

Finally, we caution the reader to not draw the erroneous conclusion that the various sources of effort being discussed can be as cleanly parsed as would appear to be indicated by the distinct boundaries drawn for each block. The reality is that rather than sharp lines, these boundaries would probably be more realistically shown as areas of different shading that blend into each other. This is particularly true for the boundary between the bottom volatility block and the larger system block.

Assessment

Assessment is the activity whereby COTS software products are vetted and selected as viable components for integration into a larger system. In general terms, viable COTS candidates are determined based on the following:

- Functional Requirements - capability offered
- Performance Requirements - timing and sizing constraints
- Nonfunctional Requirements – cost, training, installation, maintenance, reliability

The thing to remember is that COTS assessment and requirements definition must be done in concert. Final system requirements must be shaped based upon the capabilities of COTS products on the market if one is to reap the full benefit of taking the COTS approach to system development.

In the specific terms of our model, assessment is done in two passes. When selecting candidate COTS components, there is usually a quick and dirty first effort intended to very rapidly cull products patently unsuitable in the current context. The remaining products are then examined more carefully to determine the final set of COTS components to be included in the system being developed.

The key feature is that estimates are done based upon the *classes* of COTS components being examined. COTS components can usually be sorted by basic function such as GUI builders, operating systems, databases, word processors, etc. (Grouping COTS products into classes was also found by us to be the most effective way to gather calibration data. Managers and other data providers found it difficult to parse COTS integration data at either the individual component or system level, but found thinking in terms of classes of components a fairly straightforward way to ferret out the numbers for which we were looking.)

Initial Filtering Effort (IFE)=

$$\sum$$ *[(#COTS candidates in class)(average initial filtering effort for class)] over all classes*

Detailed Assessment Effort (DAE) =

$$\sum$$ *[(#COTS candidates in class)(average detailed assessment effort for class)] over all classes, by project domain*

where

the average detailed assessment effort for a given class is qualified by the attributes usually assessed for that class

Final Project Assessment Effort = IFE + DAE

The initial filtering effort formula presumes a parameterized value for a rough average filtering effort can be determined for a given class of COTS products. The initial filtering effort (IFE) for a project then becomes a simple function of the number of candidate COTS products within a class being filtered for that project, the average filtering effort per candidate within that class of products, and the total number of COTS classes represented.

The detailed assessment is similar. The parameterized average detailed assessment effort value, however, is qualified according to the kinds of attributes most often examined for a given a class of products when doing a product selection, and by the project domain (major area of

application).

Final selection of COTS products is typically based on an assessment of each product in light of certain product attributes. Table 2 contains a set of assessment attributes we've found covers most assessment cases. (Adapted from *IEEE Standards on Software Engineering*.)

Table 2 – COTS Assessment Attributes

Correctness
Availability/Robustness
Security
Product Performance
Understandability
Ease of Use
Version Compatibility
Intercomponent Compatibility
Flexibility
Installation/Upgrade Ease
Portability
Functionality
Price
Maturity
Vendor Support
Training
Vendor Concessions

Depending upon the project and its domain, more effort will be expended assessing a class of COTS product in terms of some attributes as opposed to others. For example, within one domain and class, price and ease of use may be critical, whereas in a different domain, security and portability may be paramount. Thus it follows that in the first domain, more effort will be expended assessing candidate COTS products within that class according to their purchase price and their user friendliness, while less effort (if any) will be expended assessing those same products according to the security features and portability they offer. In the second domain, the relative effort expended assessing each candidate product in terms of these four attributes should be just the reverse.

Tailoring

Tailoring is the activity whereby COTS software products are configured for use in a specific context. These are the normal things that would have to be done to a product no matter what system into which it is being integrated. These are things like parameter initialization, input/GUI screen and output report layout, security protocols set-up, etc. Specifically excluded from this definition is anything that could be considered a unique or atypical modification or expansion of the functionality of the COTS product as delivered by the vendor. (These activities would be covered under the glue code model.)

Project Tailoring Effort =

$$\sum$$ *[(#COTS tailored in class)(average tailoring effort for class and complexity)] over all classes, by project domain*

where

the average tailoring effort for a given class is qualified by the complexity of the overall tailoring task usually associated with that class

The formulation of the tailoring submodel presumes that the difficulty or complexity of the tailoring work that is going to be required to get a given COTS product integrated into a system can be anticipated and even characterized by standardized rating criteria. It presumes a parameterized value for a rough average tailoring effort for each class of COTS component, within a given project domain, can be derived based on those standard complexity ratings. The total COTS tailoring effort for a project then becomes a function of the number of COTS products within a class being tailored for that project, the average tailoring effort per component within that class of products and at the given level of complexity, and the total number of COTS classes represented.

To arrive at a complexity rating for a given tailoring job, we have defined a five-leveled scale, ranging from very low through nominal to very high. The scale, however, is really multi-dimensional, and requires examining individual tailoring activities, which affect the aggregate complexity of the job. Table 3 illustrates this concept. The first four cells in the first column under the heading "Tailoring Activities & Aids" identify the major activities that fall under our definition of COTS Tailoring, while the last cell refers to automated tools which may mitigate the difficulty of doing a given COTS tailoring job. Moving to the right across the other columns in the table you would find specific criteria (not shown) for rating the complexity of the individual activities represented by each row, or the

utility of any available tailoring tools in the case of the last row. The last column at the far right of the table provides space for recording the point values associated with the rating given to each individual item identified in the first column on the far left. These individual point values are then summed to provide the total point score indicated in the extreme lower-right corner of the table. This total score is then used to characterize the overall complexity of the given COTS tailoring effort by using table 4, titled "Final Tailoring Complexity Scale." You determine where the point total falls on the scale shown in that table and from that identify the COTS tailoring effort complexity rating associated with that point total.

Table 3 – Tailoring Complexity

Tailoring Activities & Aids	Individual Activity & Tool Aid Complexity Ratings					
	VL	L	N	H	VH	Points
Parameter Spec.	1	2	3	4	5	
Script Writing	1	2	3	4	5	
I/O Report Layout	1	2	3	4	5	
GUI Screen Spec.	1	2	3	4	5	
Security /Access Protocol Initialization & Set-up	1	2	3	4	5	
Availability of COTS Tailoring Tools	1	2	3	4	5	
						Total Points: _____

Table 4 – Final Tailoring Complexity Scale

5 - 7 pts	8 - 12 pts	13 – 17 pts	18 – 22 pts	23 – 25 pts
VL	L	N	H	VH

Glue Code

Glue code is the new code needed to get a COTS product integrated into a larger system. It can be code needed to connect a COTS component either to higher level system code, or to other COTS components also being used in the system. Reaching consensus on just what exactly constitutes glue code has not always been easy. For the purposes of our model, we finally decided on the following three part definition: glue code is software developed in-house and composed of 1) code needed to facilitate data or information exchange between the COTS component and the system or some other COTS component into which it is being integrated or to which it is being connected, 2) coded needed to connect or "hook" the COTS component into the system or some other COTS component but does not necessarily enable data exchange between the COTS component and those other elements, and 3) code needed to provide required functionality missing in the COTS component and which depends upon or must interact with the COTS component.

Glue Code Effort =

$$A \bullet [(size)(1+CREVOL)]^{B} \bullet \prod(\text{effort multipliers})$$

where

- *A = linear scaling constant*
- *Size = size of the glue code in source-lines-of-code or function points*
- *CREVOL = percentage rework of the glue code due to requirements change or volatility in the COTS products*
- *B = an architectural nonlinear scaling factor*
- *Effort multipliers = 13 multiplicative effort adjustment factors with ratings from very low to very high*

The formulation of the glue code submodel uses the same general form as does COCOMO, but defines a different set of effort multipliers. The model presumes that the total amount of glue code to be written for a project can be predicted and quantified (in either source lines of code or

function points), including the amount of reworking that code will likely undergo due to changes in requirements or new releases of COTS components by the vendors during the integration period.

Also presumed to be predictable are the broad conditions that will obtain while that glue code is being written--in terms of personnel, product, system, and architectural issues--and that these too can be characterized by standardized rating criteria. The model then presumes that a parameterized value for a linear scaling constant (in either units of person-months per source lines of code or person-months per function points) can be determined for a given domain (this is the constant A in the formula).

Table 5. Glue Code Effort Multipliers

Personnel Drivers
1) ACIEP - COTS Integrator Experience with Product
2) ACIPC - COTS Integrator Personnel Capability
3) AXCIP - Integrator Experience with COTS Integration Processes
4) APCON - Integrator Personnel Continuity
COTS Component Drivers
5) ACPMT - COTS Product Maturity
6) ACSEW - COTS Supplier Product Extension Willingness
7) APCPX - COTS Product Interface Complexity
8) ACPPS - COTS Supplier Product Support
9) ACPTD - COTS Supplier Provided Training and Documentation
Application/System Drivers
10) ACREL - Constraints on Application System/Subsystem Reliability
11) AACPX - Application Interface Complexity
12) ACPER - Constraints on COTS Technical Performance
13) ASPRT - Application System Portability
Nonlinear Scale Factor
AAREN - Application Architectural Engineering

Other parameterized values are also presumed to be determinable by domain: a nonlinear scale factor that accounts for diseconomies of scale that can occur depending on how thoroughly the architecting of the overall system into which the COTS component is being integrated was conducted; and some thirteen *effort adjustment factors* that linearly inflate or deflate the estimated size of the glue code writing effort based upon a rating from very low to very high of specific project conditions.

The total glue code writing effort for a project then becomes a function of the amount (or size) of glue code to be written, its estimated percentage rework, the linear constant A, the rated nonlinear architectural scale factor, and the individual rated effort adjustment factors.

Table 5 lists the thirteen linear effort multipliers and one nonlinear scale factor that currently appear in the model. Each of these factors has detailed definitions and rating criteria, and the interested reader is referred to the COCOTS website [9] for complete descriptions of these factors.

System Volatility

System Volatility effort refers to that extra effort which occurs during the development of the larger application as a result of the use of COTS components in that system development. Specifically, it is the effort that results from the impact on the larger system of the effects of swapping COTS components out of the system with newer versions of those components that have been released by the COTS vendors. Over a lengthy initial development these impacts can be significant. Once the project has moved into the long-term maintenance phase, these impacts will be significant, and may in fact dominate your system maintenance processes, depending upon the number and nature of the COTS components you have used in your system.

The volatility submodel tries to capture these effects by again borrowing from COCOMO the concept of rework due to factors other than errors in coding. In COCOMO, this is defined to be the amount of rework that must be done to code not because of errors in the initial programming, but rather because of changes in system requirements.

In the glue code submodel, we added the additional qualifier of rework that must be done not just due to requirements change, but also as a result of integrating upgraded versions of a COTS component. This gave us the term CREVOL.

The volatility submodel considers two other kinds of rework: that in the *larger system application* code due to integration of updated COTS software versions (SCREVOL), and that in the application code independent of COTS product effects. (This latter rework is actually the same as defined within COCOMO itself and captured in the COCOMO REVL, or *requirements evolution* term.)

System Volatility Effort =

(application effort) • {[1+(SCREVOL/1+REVL)]^{E} –1} • (COTS effort multipliers)

where

- *Application effort = new coding effort separate from COTS integration effects*
- *SCREVOL = percentage rework in the system due to COTS volatility and COTS requirements change*
- *REVL = percentage rework in the system independent of COTS effects due to requirements change*
- *E = 1.01 + Σ (COCOMO Scale Factors)*
- *COTS effort multipliers = 13 multiplicative effort adjustment factors from the glue code submodel*

The application effort term appearing in this equation is that effort that would be modeled directly by COCOMO II. The effort adjustment factors, however, are those that appear in the COCOTS glue code submodel. Finally, the scale factor term appearing in the equation are the COCOMO II scale factors (*Precedentedness, Development Flexibility, Architecture/Risk Resolution, Team Cohesion, and Process Maturity*—for a complete discussion of these terms, see references [1 & 2]).

Total COTS Integration Effort

At the moment, COCOTS is treating this as the linear sum of the four sources of effort:

Total COTS Integration Effort =

Assessment Effort + Tailoring Effort + Glue Code Effort + System Volatility Effort

As an approximation, this works for now, but the true relationship between this terms is likely more complicated.

4 CALIBRATION DATABASE OVERVIEW

We currently have 20 datapoints (information on historical industrial software projects using COTS components) in our possession, and our data collection efforts are ongoing. This data is being used to calibrate the parameters in our model.

The distribution of datapoints across various project domains (numbers are approximate):

- Air Traffic Management (50%)
- Business (including databases) (25%)
- Comm/Navigation/Surveillance (25%)
- Mission Planning (<2%)
- Logistics (<1%)

The classes of COTS products found in these projects:

- Databases
- Data conversion packages
- GUIs
- Operating Systems
- Network managers
- Device drivers
- Report generators
- Back office/retail

The above demonstrates that we have the beginnings of a varied cross section of domains and products, which is desirable when attempting a general calibration of an estimation model. Also, nearly all of the projects went through initial delivery within the last 3 years, so the data is recent, which is also helpful; i.e., it should reflect current practice.

5 CURRENT PREDICTIVE ACCURACY

As of this writing, we've had the opportunity to do a preliminary calibration on only one of our four submodels, the glue code submodel. Moreover, that calibration was based on a subset of some 13 datapoints that had actually been transcribed into our electronic database in time for the analysis. The results of that calibration are presented here.

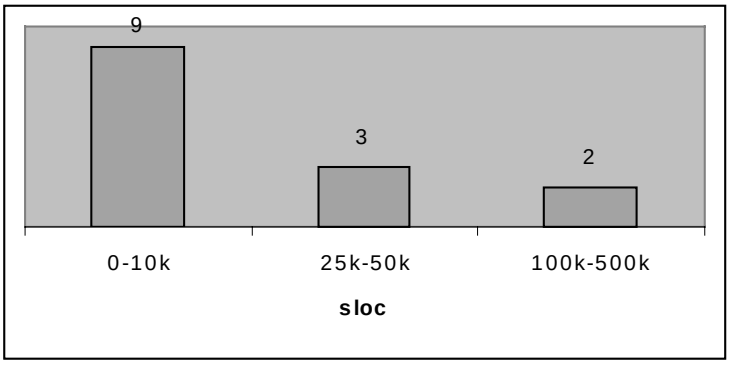

Figure 3. Glue code size in source lines of code as reported by number of projects (14 in this case)

Glue Code Effort Prediction
Based on 13 points the glue code submodel came within:

- 50% of actuals 62% of the time
- 33% of actuals 38% of the time

(For comparison, when it debuted in 1997 calibrated to 83 datapoints, COCOMO II came within 30% of actuals 52% of the time [10].)

Glue Code Schedule Prediction
Based on 13 points the glue code submodel came within:

- 31% of actuals 54% of the time

(Again, COCOMO II.1997 at 83 datapoints: 30% of actuals 61% of the time [10].)

These results are more encouraging than they appear, because what is *not* shown by these numbers but is revealed by a closer reading of the actual calculation results is that the glue code submodel was relatively well-behaved. The model tended to overestimate as often as underestimate, and generally by almost equal percentages at the extremes. The major exception to the latter, a project showing a relative error nearly twice as large as any of the others, also happened to be the one project reporting the greatest amount of glue code, nearly three times as much as any other project. It stands to reason that the larger the project, the wider potential margin for error. This also was a project with an unusually long development life, with the attendant personnel turnover and changing environmental conditions to which such developments are subject, so the impact of noise in this data point also cannot be discounted. At the very least, we feel the data overall indicate that we seem to be on the right track and should expect the predictive results to improve as the number of calibration points increases.

6 EFFORTS TO EXTEND THE MODEL

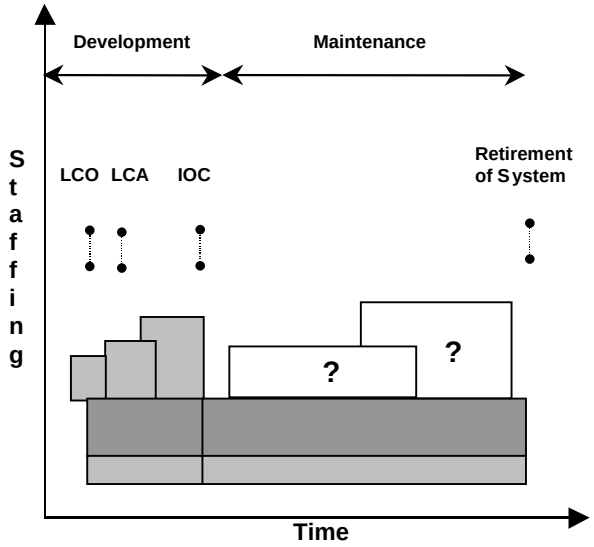

Figure 4. Expanding COCOTS for total lifecycle

Up to this point, we have been discussing COCOTS in terms of COTS integration as part of initial development. But the real impact of using a COTS approach to building software systems is on the back end, during maintenance. This is where volatility effects become most apparent and present the biggest challenges. Over a long enough planning horizon, maintenance costs for any software system--let alone a COTS-based system--will far outstrip those of development. Being able to predict those costs for systems that do use COTS is thus important.

To that end, while continuing data collection for and refinement of the version of the model we already have, we have also begun exploration of the factors which would be most useful in modeling the costs of COTS-based systems all the way through system retirement. As yet unknown and symbolized by the boxes with question marks in figure 4, this will provide a rich area for future work.

ACKNOWLEDGEMENTS
We would like to thank the following groups for supporting the development of COCOTS: the USAF, the FAA, the Office of Naval Research, the SEI, the USC-CSE Affiliates, the members of the COCOMO II research group, and most especially the organizations and individuals that have generously supplied us with data but must remain unnamed.

REFERENCES

1. Boehm, B., Clark, B., Horowitz, E., Madachy, R., Shelby, R. and Westland, C., "Cost Models for Future Software Lifecycle Processes: COCOMO 2.0," in *Annals of Software Engineering*, 1995.
2. COCOMO II Model Definition Manual, 1998. At http://sunset.usc.edu/COCOMOII/cocomo.html.
3. adapted from Boehm, B. and Abts, C., "COTS Integration: Plug and Pray?," *Computer*, Jan. 1996, pp. 135-138.
4. Boehm, B. Software Engineering Economics, *Prentice Hall*, NJ, 1981.
5. Chulani, S., Boehm, B., and Steece, B., "Calibrating Software Cost Models Using Bayesian Analysis," *Proceedings 1999 ISPA/SCEA Conference*.
6. Chulani, S., "Incorporating Bayesian Analysis to Improve the Accuracy of COCOMO II and its Quality Model Extension," USC-CSE tech. report 98-506.
7. Abts, C. and Boehm, B., "COTS Software Integration Cost Modeling Study," USC-CSE tech. Report 98-520.
8. Boehm, B., "Anchoring the Software Process," *IEEE Software*, July 1996, pp. 73-72.
9. COCOTS website: http://sunset.usc.edu/COCOTS/cocots.html.
10. Chulani, S., Clark, B., Boehm, B., Steece, B., "Calibration Approach and Results of the COCOMO II Post-architecture Model," *Proceedings 1998 ISPA Conference*.