NOTE TO USERS

Page(s) not included in the original manuscript and are unavailable from the author or university. The manuscript was scanned as received.

167-190

This reproduction is the best copy available.
EXTENDING THE COCOMO II SOFTWARE COST MODEL TO ESTIMATE EFFORT AND SCHEDULE FOR SOFTWARE SYSTEMS USING COMMERCIAL-OFF-THE-SHELF (COTS) SOFTWARE COMPONENTS: THE COCOTS MODEL

by

Christopher Mark Abts

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(INDUSTRIAL AND SYSTEMS ENGINEERING)

May 2004

Copyright 2004
Christopher Mark Abts
Acknowledgements

This has been a (very) long time coming and many people have offered me support, encouragement and assistance along the way. These include family, friends, academic mentors, and professional colleagues. Rather than explicitly thank a few when I am actually indebted to so many, I’ll simply offer here publicly my heartfelt gratitude to all, with the hope that in the days and weeks ahead I’ll be able to offer my true thanks to each of you in person.

However, I do have to offer a particular and profound thank you to the members of my dissertation committee, for their guidance as well as their patience:

- Dr. F. Stan Settles, Dept. of Industrial & Systems Engineering (committee chair)
- Dr. Satish T. S. Bukkapatnam, Dept. of Industrial & Systems Engineering
- Dr. Barry W. Boehm, Dept. of Computer Science (outside member)
Table of Contents

Acknowledgements ii

List of Tables vi

List of Figures vii

Abstract ix

1. Introduction 1
   1.1. COCOMO and the Evolution of Modern Software Development Practice 1
   1.2. The COTS-Based System (CBS) Approach to Software Development 10
   1.3. Thesis Topic: Extending COCOMO II to Model a CBS Development Process (COCOTS) 17
   1.4. Organizations Supporting this Research 19
   1.5. Chapter 1 Endnotes 21

2. Background and Related Work 23
   2.1. State of the Art vs. State of the Practice 23
   2.2. Overview of Software Cost Estimation Methodologies 28
       2.2.1. Expert Opinion 29
       2.2.2. Case Studies & Analogy 33
       2.2.3. Top Down & Bottom Up 35
       2.2.4. Statistical Regression 38
       2.2.5. Parametric Modeling 44
       2.2.6. Neural Networks 49
       2.2.7. Dynamic Simulation 51
       2.2.8. Bayesian Analysis 56
   2.3. Existing Parametric COTS-related Cost Estimation Models 61
       2.3.1. Stutzke Model 61
       2.3.2. Alternate SAIC Model 62
       2.3.3. Ellis Model 63
       2.3.4. Price-S 67
       2.3.5. SEER-SEM 68
   2.4. Description of COCOMO II 72
       2.4.1. Effort Estimation 72
       2.4.2. Schedule Estimation 86
   2.5. Chapter 2 Endnotes 89
3. Research Approach
  3.1. Setting the Context
  3.2. USC-CSE Seven Step Modeling Methodology
    3.2.1. Steps 1 through 3: Preliminary Model Definition
    3.2.2. Step 4: Delphi Experiment
    3.2.3. Step 5: Data Collection & Conditioning
    3.2.4. Step 6: Bayesian-assisted Model Calibration
    3.2.5. Step 7: Model Refinement & Renewed Data Collection
  3.3. Chapter 3 Endnotes

4. Research Contributions
  4.1. COCOTS Development Phase Effort Model Calibration
    4.1.1. First Iteration: Initial COTS Integration Cost Model Study
    4.1.2. Second Iteration: Initial FAA Revision
    4.1.3. Third Iteration: Second FAA/ONR Revision
    4.1.4. Fourth Iteration
  4.2. Discussion and Conclusions
  4.3. Chapter 4 Endnotes

5. Areas for Future Work
  5.1. Schedule Modeling
  5.2. CBS Maintenance Modeling from IOC through Retirement
    5.2.1. Maintenance Scheduling as an Integer Programming Problem
  5.3. Other Areas for Investigation
    5.3.1. Risk Assessment
    5.3.2. Project Management
    5.3.3. CAIV and SAIV Modeling
    5.3.4. Modeling the use of WWW-based COTS Applications
    5.3.5. The COTS-LIMO Model and End-of-Service-Life Determination
    5.3.6. CBS Functional Density Rule and CFD Metric
    5.3.7. COTS-based Function Points
    5.3.8. CBS-oriented Work Breakdown Structure
    5.3.9. COTS Product Assessment and Tailoring Profiles
  5.4. Chapter 5 Endnotes

6. Bibliography
Appendices

A  Fourth Iteration COCOTS Model Parameter Values  202
B  COCOTS Data Collection Survey  203
C  Summaries of Collected COTS-based Systems Data  207
D  Guidelines for Using COCOTS with COCOMO II  259

II
List of Tables

1.1 – COTS Advantages and Disadvantages 14
2.1a – COCOMO II.2000 Scale Driver Rating Criteria 80
2.1b - COCOMO II.2000 Post-Architecture Effort Multiplier Rating Criteria 81
2.1c – CPLX Rating Criteria 82
2.2 – COCOMO II.2000 Scale Factor and Effort Multiplier Numeric Values 84
2.3 – Predictive Accuracy of COCOMO II.2000 85
3.1 – Predictive Accuracy of Three Calibration Approaches for COCOMO II 113
4.1 – First Pass Model Parameters 127
4.2 – Second Pass Model Parameters 134
4.3 – COTS Assessment Attributes 145
4.4 – Third Pass Glue Code Submodel Parameters 149
4.5 – Fourth Pass Glue Code Submodel Parameters 166
List of Figures

1.1 – a traditional waterfall engineering development process model 3
1.2 – a spiral engineering development process model 5
1.3 – the determinants of a feasible COTS-based system solution 16
2.1 – the spectrum of software-related estimation techniques 28
2.2 – a product work breakdown structure 38
2.3 – an activity work breakdown structure 38
2.4 – a regression line fitted through empirical data so as to minimize $r^2_{i}$ across all observations 40
2.5 – a regression line skewed by an extreme outlier 42
2.6 – the Norden-Rayleigh distribution of labor over the software project life cycle 47
2.7 – a neural network estimation model 50
2.8 – Madachy’s system dynamics model of Brooks’ Law 53
2.9 – influence of variance on a given estimated value determined by Bayesian statistical methods 58
2.10 – SEER-SEM inputs and outputs 70
3.1 – USC-CSE modeling methodology 96
3.2 – COTS component role being addressed by COCOTS: COTS as application elements 102
4.1 – first pass COTS estimation model 122
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>COTS assessment activity in the CBS development life cycle</td>
<td>125</td>
</tr>
<tr>
<td>4.3</td>
<td>second pass COTS estimation model</td>
<td>132</td>
</tr>
<tr>
<td>4.4</td>
<td>third pass COTS estimation model</td>
<td>140</td>
</tr>
<tr>
<td>4.5</td>
<td>fourth pass COTS estimation model</td>
<td>160</td>
</tr>
<tr>
<td>5.1</td>
<td>CBS post-deployment modeling horizon</td>
<td>179</td>
</tr>
<tr>
<td>5.2</td>
<td>customer base dependent on CBS field date</td>
<td>184</td>
</tr>
<tr>
<td>5.3</td>
<td>the COTS-LIMO model</td>
<td>186</td>
</tr>
<tr>
<td>D.1</td>
<td>COTS-based system components mapped to appropriate estimation model</td>
<td>281</td>
</tr>
<tr>
<td>D.2</td>
<td>a hypothetical COTS-based system architecture</td>
<td>282</td>
</tr>
<tr>
<td>D.3</td>
<td>life cycle phases covered by COCOMO II and COCOTS</td>
<td>286</td>
</tr>
</tbody>
</table>
Abstract

Since the inception of the field some fifty years ago, the way in which computer software is developed has remained fluid, continually evolving new techniques in the never ending quest to produce software more quickly, more cheaply, and of ever higher quality. A consequence of this continual evolution in software development techniques is that estimation models devised to predict the effort and schedule likely needed to build software systems grow less reliable and useful if the assumptions about the development practices upon which these models are formulated are not themselves updated.

This dissertation discusses changes in software development practice that have occurred since the 1970s. In particular, the focus is on the rise of the "off-the-shelf" approach whereby large software systems are constructed out of a mix of original code and adapted code as well as pre-existing "black box" components for which the developer has no access to the source code. While certainly used prior to 1980, this technique of using black box elements has taken on greatly increased importance in recent years as one way to manage development risks as the overall complexity of software has grown.

Software estimation models, however, have generally not kept pace with off-the-shelf techniques. This includes the widely-used COCOMO software estimation model. While recently updated as the COCOMO II estimation model to reflect other
changes in software development since its original publication in 1981, it still is unable to accommodate the use of black box off-the-shelf components in its estimation framework.

The focus of this dissertation then is the investigation of a potential extension of COCOMO II that is intended to provide an estimation capability for any off-the-shelf elements being designed into a software system. It begins with an exploration of how development activities differ when using off-the-shelf components as opposed to developing system components as original code. This serves as a basis for the formulation of an off-the-shelf estimation model called COCOTS that is carried through several iterations, each intended to improve upon its predecessor. The dissertation concludes with a comparison of the fidelity of COCOTS with COCOMO II, limitations of available project data that impacted the calibration of COCOTS, and suggestions for further investigation that might improve the fidelity of COCOTS itself and also expand and deepen the general understanding of the benefits and risks of using off-the-shelf components in software system development.
1. Introduction

1.1 COCOMO and the Evolution of Modern Software Development Practice

In the field of software engineering, cost estimation models that attempt to predict the effort and schedule that will be required to build new software systems are used for a variety of purposes.\(^1\) The most obvious of these are budgeting and scheduling, but these are far from their only utility. An equally important use would include aiding in software investment decisions—for example, whether to develop needed software functionality from scratch, or by reusing or modifying existing software to achieve that same functionality, or even perhaps to purchase the needed functionality outright from a commercial vendor. Still other uses for software cost estimation models include working out software cost vs. completion schedule vs. quality vs. functionality tradeoffs; performing analyses of software project risks and their attendant mitigation strategies; doing analyses of investment in software development process improvement; and even making capital equipment maintenance and replacement decisions where "capital equipment" in this case refers to software—for example, Enterprise Resource Planning (ERP) systems, which are large scale computing systems typically used corporate-wide and which demand significant capital investments.

In short, the purpose of these models is "to help people reason about the cost and schedule implications of their software decisions."\(^2\)
COCOMO (acronym for COnstructive COst MOdel) is one such software cost estimation model. Specifically, it is a planning aid that allows one to estimate the cost, effort, and schedule associated with a given software system design (or designs), both during the initial stages of a new software development activity as well as during the latter life of such a system after it has gone into maintenance. Originally published in 1981, it was one of the first such empirically-based models to appear in the literature, and eventually became (and still remains) one of the most widely-used parametric cost estimation models in the software industry. One of the reasons for this is its completely transparent nature, with its internal equations and parameter values openly published and available to all. (This is why it's called a "constructive" model, because you can see explicitly how its estimate is developed, helping you to better understand how all the pieces of the software job to be done fit together—in other words, it offers constructive insight into the task at hand.)

COCOMO 81 (as the original model is now officially known, but which is also on occasion referred to by some as COCOMO I) was developed during the latter half of the 1970s, and naturally reflected the software development practices of the day. Software during this period was typically built from scratch, using the tried and true "waterfall" process approach long used by architects, civil engineers, and hardware designers. This process carries an engineering project through a clearly defined and distinct sequence of phases, from requirements definition through design, development, subcomponent integration, test and ultimately to final system delivery.
or completion (see figure 1.1). The approach is lock-step, with one phase not beginning until the preceding phase is completed or nearly so, and there is typically little feedback between and iteration on the various steps and phases. It is well-suited to situations where system requirements are well-understood early on, not subject to much revision or modification as the project progresses, and where development environment resources are significant, predictable and assured.

![Figure 1.1 - a traditional waterfall engineering development process model.](image)

Much has changed, however, in the two decades that have passed since COCOMO's first release. The environmental conditions described above and deemed desirable for a successful waterfall-based software system development frequently no longer obtain (if they ever did).

These days, even high level requirements are often unknown or at least fluid well into a software development, and can change significantly and rapidly before coding is complete in response to outside forces (e.g., the release of a competing
system or product by a market rival, either of yourself or of a vendor of any off-the-
shelf components that may be part of your system). Moreover, resources committed
to a development may be nowhere near as constant and assured as was once the case.
As just one example, project personnel turnover rates are much higher than in past
years due to the great demand for experienced software professionals in the high-
tech industry today (the recent dot "gone" experience notwithstanding).

These factors along with others have led to dramatic changes in the way
software is developed today compared to twenty years ago. Among the more
significant innovations are the following:

- a move away from mainframe overnight batch processing towards desktop-
  based real-time turnaround.
- a greatly increased emphasis on reusing existing software and building new
  systems using off-the-shelf software components.
- spending as much effort to design and manage the software development
  process as was once spent creating the software product.

Additionally, the near monolithic hold of the waterfall process model on
software system development has given way to a multitude of alternative
development approaches: reuse-driven, legacy-driven, demonstration-driven, design
to cost or schedule, incremental, evolutionary, and various hybrid combinations
thereof. Among these newer approaches, one of the more widely encompassing is
the risk-driven "spiral"\(^6\) model of system development (see figure 1.2), of which many of the other approaches can be considered special cases.

Figure 1.2 - a spiral engineering development process model.\(^7\)
The spiral approach differs from the waterfall approach in that it is cyclical rather than linear, iterating a system development through increasingly more elaborate prototypes of a proposed software system until the desired end-product is realized. It requires during each iteration a concurrent rather than sequential determination of project artifacts traditionally produced at the completion of schedule milestones. These include such items as requirements specifications, high level and detailed designs, software source code, integration test plans, roll out and delivery plans, etc. The key feature of this process model, however, is that a risk assessment done at the beginning of each prototyping cycle determines the mix of development activities and how effort and resources are to be allocated during that next cycle, repeating until the final cycle is completed and the desired product achieved.

The spiral approach is far more adaptable than the waterfall approach to situations where the initial understanding of system requirements is low and likely to change greatly before the completed system is ultimately fielded, and where it is desirable to see at least some form of the working product early on. It can also bring resources to bear in a timely fashion against unforeseen difficulties that arise during a given development in a way the waterfall approach cannot. This is because the risk analyses required before the start of each development cycle provide repeated and early opportunities for identifying and responding to such problems as they arise. Using the waterfall approach, *sans* a cyclic risk analysis, a flaw in the original
requirements specifications might not manifest itself until you've gotten all the way to the integration and test phase, at which point it will be much more difficult and expensive to address. While the same kind of requirements flaw might go unnoticed using the spiral approach, there will be more opportunities for catching it earlier rather than later during the course of the development. For this same reason the spiral approach is better suited than the waterfall approach to situations where there are a lot of unknowns or ambiguities at the start of system development beyond just the definition of the requirements.

Which brings us back to COCOMO. All these changes since the 1970s in technology and process, technique and practice, began to make applying the original COCOMO model problematic. It was having more and more difficulty reliably estimating effort and schedule for software created using these newer process approaches. The only solution to this growing problem was to reinvent COCOMO for the 21st century. Thus was born in 1994 the COCOMO research project in the Center for Software Engineering at USC. The goal of this new research effort was to examine how best to address issues of non-sequential or spiral development process models, rapid application development (RAD) process models, software reengineering and reuse-driven approaches, object oriented development, off-the-shelf development approaches, quality issues, etc., within the context of parametric predictive models. One of the prime results of this research to date has been the creation of The COCOMO Suite, a collection of several COCOMO-related
estimation models in various stages of development. These models are complimentary in nature, attempting to estimate software system development costs, development schedule, system engineering costs, and even return on technology investment in light of the broad variety of software development approaches and techniques described above.  

The premier achievement of this research to date, however, is COCOMO II itself. First described in preliminary form in 1995, it was finally published in 2000 in toto. With this new model, the problems that had become manifest over time with COCOMO 81 due to the evolution in software development techniques since its initial publication are resolved. And while it still allows you to estimate the cost, effort, and schedule associated with a software system constructed using new and reuse components, it does so at three levels of increasing fidelity, depending on how far along you are in the project planning and design process. Listed in increasing fidelity, the COCOMO II submodels that offer this capability are called the Applications Composition, Early Design, and Post-architecture models.

The Application Composition model is used to estimate effort and schedule on projects that typically use Integrated Computer Aided Software Engineering (ICASE) tools for rapid application development. Projects of this nature can be highly varied in terms of their domains but share the characteristic of being sufficiently simple to be rapidly composed from interoperable components. These components typically are items such as Graphical User Interface (GUI) builders,
database or objects managers, middleware for distributed processing or transaction processing, and domain-specific components such as financial, medical or industrial process control packages.\textsuperscript{12}

The Early Design model involves the exploration of alternative system architectures and concepts of operation. At this stage of project planning, there is usually not enough information to make a detailed effort and schedule estimate, but you do have some idea as to likely environmental project conditions.

The Post-Architecture model is used when top level design is complete and the software architecture is well defined and established. At this stage of project planning, there is now enough known about the project to attempt a detailed effort and schedule estimate.

The Post-Architecture model is the most elaborate of the three COCOMO II submodels (see section 2.4) and to date has been calibrated using a Bayesian statistical approach to a database of 161 historical industrial software development projects.\textsuperscript{13} These projects were collected from a mix of commercial, aerospace, government and non-profit organizations. Across all data points, the Post-Architecture model as currently calibrated consistently provides effort estimates within 30\% of actual reported effort about 75\% of the time. After segregating the data by reporting organization, the results improve to being within 30\% of actuals some 80\% of the time.
The current calibration for the Early Design model was obtained by aggregating calibration numbers determined for the Post-Architecture model, but sufficient Early Design historical project data has yet to be collected to produce statistically meaningful independent measures of the Early Design model's accuracy.\textsuperscript{14}

Meanwhile, the Application Composition model has been calibrated to data published by Robert Kauffman and Rachna Kumar, but the results were such (within 30\% of reported actuals less than 50\% of the time) that this model is still considered experimental.\textsuperscript{15}

For all that it does, however, and all the calibration data that has been collected, what COCOMO II does \textit{not} model is the situation in which some parts of a new software system are to be composed using black box off-the-shelf components. Since this approach is becoming more and more the norm when developing new systems, the inability to model this scenario represents a serious gap in the capability offered by COCOMO II.

Filling in this gap provided the fundamental motivation for the doctoral research presented in this report.

1.2 The COTS-based System (CBS) Approach to Software Development

One of the more significant changes in the software development field over the past twenty years is the greatly increased emphasis being placed on building systems
incorporating pre-existing software, with special emphasis being placed upon the use of commercial-off-the-shelf (COTS) software components. This is especially true with respect to software systems being purchased by the United States federal government, most notably within the Department of Defense. In the 1990s, new DoD procurement policies increasingly called for mandated levels of COTS component use. In 1993, the U.S. Navy supposedly even went so far as to establish a policy stating that the selection of a government in-house or procured software solution and not a COTS-based solution was to be taken as a rejection of a comparable COTS solution. This shift in policy meant that Navy procurers now had to justify why they were not using COTS software as part of the design of new systems.

The rationale typically given for building COTS-based systems is the belief 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 that impact the strength or weakness of this assertion in any given case—indeed, so much so that they drive the whole COTS usage process:

* There are many flavors of "off-the-shelf" (OTS) components: COTS (commercial OTS), GOTS (government OTS), ROTS ("research" OTS), GFE (government furnished equipment), NDI (non-developmental item); from a technical point-of-view (barring specific procurement and licensing issues), the term "COTS" generally can be considered to encompass all of these as long as a given item is acquired from a third party or outside organization, was not designed specifically for your use, and you have no control over how that item will evolve in the future.
the COTS product source code is not available to the application developer.*

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

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 test and integration work. The long term cost implications of adopting the COTS approach are even more profound, because from the moment you start considering COTS components for your new system up until the day you finally retire that system, you are in fact adopting a new way of doing business. Whether you realize it or not, for every unique vendor from whom you acquire COTS components, you've added a new business partner—moreover, a partner over whom you may have little influence and whose business success model may or may not coincide with your own. This is because COTS software is not static, it continually evolves in response to the vendor's market, and your business may represent only a small fraction of that market. You as the system developer must therefore adopt new methodologies not applicable to traditional system

* In some cases, vendors do supply COTS software as uncompiled source code. For the purposes of the estimation modeling effort being outlined in this report, however, if a developer does anything to that code other than compile it unexamined and unchanged into his own code, such COTS components will largely lie outside the scope of this effort. This is because such items can be treated as white box reuse components and their usage modeled as adapted code within COCOMO II itself.
developments but that nonetheless cost-effectively manage the use of these evolving components and the vendors who supply them.

The truth 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 factors that should be examined when determining the true cost of integrating a COTS software component into a larger system are these:

- the traditional costs associated with new software development such as the cost of requirements definition, design, code, test, and software maintenance.

- plus the COTS unique costs 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.

Table 1.1 further summarizes some of the more significant factors that represent possible trade-offs that may need to be considered when using COTS components.

Because of the unique risks identified under the second bullet on the previous page and the trade-offs identified in table 1.1, 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>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediately available; earlier payback</td>
<td>Licensing, intellectual property procurement delays</td>
</tr>
<tr>
<td>Avoids expensive development</td>
<td>Up-front licensing fees</td>
</tr>
<tr>
<td>Avoids expensive maintenance</td>
<td>Recurring maintenance fees</td>
</tr>
<tr>
<td>Predictable, confirmable license fees and performance</td>
<td>Reliability often unknown or inadequate; scale difficult to change</td>
</tr>
<tr>
<td>Rich functionality</td>
<td>Too-rich functionality compromises usability, performance</td>
</tr>
<tr>
<td>Broadly used, mature technologies</td>
<td>Constraints on functionality, efficiency</td>
</tr>
<tr>
<td>Frequent upgrades often anticipate organization’s needs</td>
<td>No control over upgrades and maintenance</td>
</tr>
<tr>
<td>Dedicated support organization</td>
<td>Dependence on vendor</td>
</tr>
<tr>
<td>Hardware/software independence</td>
<td>Integration not always trivial; incompatibilities among vendors</td>
</tr>
<tr>
<td>Tracks technology needs</td>
<td>Synchronizing multiple-vendor upgrades</td>
</tr>
</tbody>
</table>

COTS components are the right solution when they lie at the intersection (see figure 1.3) of the three determinants of feasibility—technical, economic, and strategic constraints—and do so in a way demonstrably better than if a new system were to be constructed entirely out of original software. The key to success in using
COTS components is being able to identify whether they fit the current procurement situation—technically, economically, and strategically. Technically, they have to be able to supply the desired functionality at the required level of reliability. Economically, they have to be able to be incorporated and maintained in the new system within the available budget and schedule. Strategically, they have to meet the needs of the system operating environment—which includes technical, political, and legal considerations—now, and as that environment is expected to evolve in the future.

Technical and strategic feasibility is determined during the candidate assessment phase of procuring COTS products, which occurs at the start of a COTS integration activity. How to determine the viability of a COTS product in either of these two dimensions is not a trivial question, and will remain for most part (though not entirely) outside the scope of this research. It is the third dimension of determining economic feasibility that is the primary focus here.
To answer the question of economic feasibility, cost estimation models exist which capture the traditional costs noted above that are associated with new software development, COCOMO itself being among the most prominent of these models. A review of the literature, however, both in traditional publications and on the internet, suggests that to date, very few estimation models have been developed which try to capture those other costs unique to using COTS components in a software system development. The number of COTS-oriented cost models available in the public domain is even fewer. Hence the effort undertaken to extend COCOMO II to cover

---

* Examples of software estimation models other than COCOMO include Checkpoint, SLIM, PRICE-S, SEER-SEM, Estimacs, Softcost; there are others, including a host of models derived from COCOMO itself: CoCoPro, COCOMOID, COSTMODL, COST^XPERT, REVIC, and COSTAR. Information on most of these models can be found by doing searches on the internet, but a couple of them (SLIM, PRICE-S, SEER-SEM) will be discussed later in this report.
the development of COTS-based systems was expected to provide a valuable contribution to the field.

1.3 Thesis Topic: Extending COCOMO II to Model a CBS Development Process (COCOTS)

As indicated above, COCOMO I creates effort and schedule estimates for software systems built either from scratch (that is, new code), or from pre-existing source code that is modified or adapted to the current purpose (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 the new cost model being described here comes in. It is not being proposed necessarily to stand on its own so much as to serve as a complimentary extension of the existing COCOMO II model. Moreover, in keeping with precedent and to highlight that complimentary relationship to its parent model, the proposed new extension model is being dubbed COCOTS, short for COstructive COTS model.
The ultimate goal is to have a single unified tool implementing algorithms that allow you to estimate cost and schedule for the modern COTS-based system. In the typical contemporary CBS, the new software system is constructed out of a mix of components: new code modules, white box reuse or adapted code modules, and black box off-the-shelf modules. At the moment, COCOMO can handle the first two items. The new extension, is intended to add the capability to model the third kind of item.

There are challenges to doing this, however. COCOMO II already has a well-established credibility within industry. Any extension must do nothing to diminish faith in what the model can do currently. To avoid this risk, the existing COCOMO effort modeling algorithms are thus being treated as sacrosanct. The new COCOTS effort modeling algorithms were developed independently from the main model, but will be used in conjunction with what is already there (see section 2.4.1). The intersection of the two effort models will (eventually) occur in the COCOMO II schedule algorithm (see section 2.4.2). The schedule algorithm currently takes as its primary input the effort estimate produced by COCOMO. However, this only covers system elements built from scratch or from reuse components. To determine a completion schedule for the entire CBS in which off-the-shelf elements are added to the mix, COCOTS effort estimates will in the future need to be combined in some fashion with COCOMO effort estimates to get a unified schedule algorithm.
This then becomes the central hypothesis to be investigated with this thesis:

Is it possible to create an expanded COCOMO-based parametric model that will estimate the initial development effort for software systems composed of a mix of original, adapted, and off-the-shelf elements to the same fidelity as COCOMO II.1997* estimates the initial development effort for software systems composed of original and adapted software elements alone?

1.4 Organizations Supporting this Research

It was stated in section 1.2 that the US federal government places much importance in the ever-increasing role of COTS components in the software systems currently being procured by the military and other government departments. As a measure of that consensus, it is useful to note that several government organizations have already supplied significant funding to support this research into CBS modeling.

The first grant came from the USAF Electronics Systems Center at Hanscom AFB in Massachusetts, which funded my preliminary COTS modeling efforts described in a COTS modeling study concluded in 1997 (see chapter endnote #18).

Subsequent to that, the Federal Aviation Administration (FAA) provided funding for the next phase of the research, which allowed a refinement of the first CBS model described in the 1997 study, and the initiation of a new round of CBS data collection20.

* The established naming convention for COCOMO II identifies a specific calibration, thus COCOMO II.1997 refers to the 1997 calibration based on 83 historical data points while COCOMO II.2000 refers to the 2000 calibration based on 161 data points.
About the same time the FAA support was realized, the Office of Naval Research (ONR) also provided funding for this research, which helped with yet another, even more significant revision of the model\textsuperscript{21}.

As of this writing the FAA support is still ongoing, and that organization is continuing to provide significant assistance in the collection of CBS data for modeling purposes.

Finally, as affiliates of the USC Center for Software Engineering (the university research unit out of which I have been conducting these studies), the companies and organizations listed below have also provided important support for this effort, not only in terms of funding, but also by their willingness to make available their resident CBS experts for consultation and participation in certain aspects of this work.

\textit{Aerospace Corp.}
\textit{Automobile Club of Southern California}
\textit{Boeing Corp.}
\textit{C-Bridge}
\textit{Chung - Ang University, Korea}
\textit{Draper Labs}
\textit{Electronic Data Systems}
\textit{Federal Aviation Administration}
\textit{Fidelity}
\textit{GDE Systems}
\textit{Group Systems,Com}
\textit{Hughes Aircraft Company}
\textit{IBM, Inc.}
\textit{Institute for Defense Analysis}
\textit{Jet Propulsion Laboratory}
\textit{Litton Data Systems}
\textit{Lockheed Martin Corp.}
\textit{Lucent}
\textit{Microsoft}
\textit{Motorola}
\textit{Northrop Grumman Corp.}
\textit{Rational, Inc.}
\textit{Raytheon/East}
\textit{Raytheon/West}
\textit{Science Applications International Corp.}
\textit{Software Engineering Institute}
\textit{Software Productivity Consortium}
\textit{Sun Microsystems}
\textit{Telcordia}
\textit{TRW}
\textit{U.S. Air Force Rome Laboratory}
\textit{U.S. Army Research Laboratory}
\textit{U.S. Army TACOM}
\textit{Xerox}
1.5 Chapter 1 Endnotes


2 Boehm, Barry, TRW Professor of Software Engineering, University of Southern California, Los Angeles, CA.


7 Figure reproduced from Boehm, B., "A Spiral Model of Software Development and Enhancement," PowerPoint briefing, University of Southern California, Los Angeles, CA, September 15, 1998.

8 Internet home page for USC-CSE is at http://sunset.usc.edu.

9 Internet page for The COCOMO Suite is at http://sunset.usc.edu/research/cocomosuite/index.html.


19 Figure reproduced from Abts and Boehm, "COTS Software Integration Cost Modeling Study," p. 3.


2. Background and Related Work

2.1 State of the Art vs. State of the Practice

An examination of the public literature reveals an interesting phenomenon. There has been much discussion in recent years about the topic of COTS integration, but most of it has been restricted to framing the issue in qualitative terms. Quantitative COTS integration models available for review in the public domain are almost non-existent. However, in terms of the former, several good offerings help to define key issues of concern pertaining to COTS integration, flagging risks in using COTS software and suggesting strategies for avoiding or mitigating those risks.

In the interests of keeping my thesis topic bounded, discussions along these lines have remained largely (though again not entirely) outside of the scope of the empirical research being described here. An understanding of these issues, however, is important in the larger context of successfully applying any CBS cost model that might be developed, whether it be COCOTS or some other model. The reason is that no parametric software development cost model exists in a vacuum. To intelligently interpret any estimates such a model might provide demands an understanding of 1) the assumptions underlying the structure and inputs to the model, 2) the forces that impact the design and evolution of the software system under consideration, and 3) the choices available to project leaders in response to what any given estimate might suggest about the current state of those aforementioned forces affecting the development project. In light of this, some representative sources of qualitative
information regarding the use of COTS components in software systems are offered below.

The COTS-Based Systems Initiative\(^1\) sponsored by the Software Engineering Institute (SEI) at Carnegie Mellon University is a very significant resource for those interested in learning about the issues surrounding the use of COTS components. In particular they offer a series of monographs addressing the following areas:\(^2\)

- finding and selecting appropriate commercial products.\(^3\)
- identifying decision criteria for migrating to new or emerging COTS technologies.
- understanding the ramifications of the CBS approach on system architecture.
- developing testing strategies for systems incorporating COTS components.

None of the SEI monographs directly broach the topic of CBS cost estimation, but cost considerations certainly play a role in each of the areas noted above.

The Software Productivity Consortium\(^4\) (SPC) headquartered in Herndon, Virginia also offers a well thought-out process for COTS software component evaluation and selection to both its member companies and organizations as well as to industry at large. Their process addresses the following:\(^5\)
scoping COTS component evaluation efforts.

- searching and screening candidate COTS components.

- defining the appropriate criteria for evaluating COTS components.

- gathering evaluation data on various alternative COTS components.

- analyzing the results of the evaluation data gathering activities.

Another source of highly useful qualitative information on building software systems with COTS products is the growing body of "lessons learned" reports and briefings that are slowly becoming available as existing CBS developments mature and move from the initial development to the long term maintenance phase. Again, these documents rarely dwell on specific cost estimation techniques. Rather, they tend to provide insight into good CBS acquisition practice as well as guidance regarding effective strategies to employ in order to mitigate many of the unique COTS-specific risks (see section 1.2) that can threaten the success of a CBS development. A constructive CBS parametric cost model would go a long way to furthering the successful implementation of many of these practices and risk mitigation strategies. Examples of four such "lessons learned" documents are the following:

* It should be noted that in this area in particular, the SEI has worked in cooperation with the National Research Council Canada (NRC) to define an extensive process for evaluation and selection of COTS products (see chapter endnote #3).
• a Federal Aviation Administration (FAA) report\textsuperscript{6} that is based directly on findings derived from more than 200 hours of data collection interviews already conducted to date by myself and another researcher\textsuperscript{*} in support of this COTS modeling effort; it provides in simple language a distillation of quotes and insights gathered from across twenty recent large CBS developments residing mostly (though not exclusively) in the aviation domain (see end reference #27).

• a Department of Defense (DoD) report\textsuperscript{7} prepared by the SEI that focuses on the areas of commercial business practices, the evaluation and selection of COTS products, managing relationships with COTS vendors, and CBS design practices conducive to long term life cycle support; it has a DoD acquisition officer bias but still provides many useful insights into managing a CBS development.

• a Lockheed Martin briefing\textsuperscript{8} outlining a specific CBS acquisition, COTS product selection and COTS component integration and life cycle support process based on experiences with at least a half-dozen large CBS developments in the aerospace domain; it offers extensive guidance on

\textsuperscript{*} Dr. Elizabeth (Betsy) Clark of Software Metrics, Inc., Haymarket, VA, has been my partner in lining up data collection opportunities for this research and in refining certain aspects of the proposed COCOTS model. Her assistance has been invaluable with regard to ensuring the empirical foundations of this research.
implementing a successful CBS development process as well as detailed rationale for assertions made regarding learned good practice.

- another SEI monograph focusing on areas of general COTS precepts, CBS requirements, COTS product evaluation and CBS design, CBS maintenance, COTS vendor and procurer business processes, CBS testing and correction, and the role of Integrated Product Teams (IPTs) in the context of a CBS development; it takes a whimsical yet informative approach to the subject matter, presented in a format mimicking Mao Tse-tung's famous Little Red Book.

The above items should not be construed as exhaustive, only representative of the kind of "best practices" information that is slowly becoming available regarding CBS developments, knowledge that is helping to inform both the development and application of COCOTS.

Now that the state of available qualitative COTS related information has been discussed, the next two sections of this document will focus on two topics more directly related to the development of a CBS cost estimation model. The first topic is estimation methodologies that have been employed previously on traditional, non-COTS-based software system developments. The second topic is a review of the (very) few extant CBS cost estimation models found in the literature.
2.2 Overview of Software Cost Estimation Methodologies*

The history of empirically supported reasoning in software cost estimation goes back some 35 years, beginning with a Systems Development Corporation (SDC) study in 1966 of the attributes characterizing nearly 170 software development projects. Over 100 project attributes were identified by this study, leading to some simple estimation models used in the late 1960s and early 1970s.

* This section is adapted and expanded from chapter 1 endnote #12, "Software Development Cost Estimation Approaches - A Survey," by Boehm, Abts, & Chulani, and Chapter 22 of chapter 1
But the general history of software estimation methodologies goes back even further, to the birth of software as an industry in the 1950s. In those early days, software project estimates were based mainly on the expertise of senior individuals who had seen it all and done it all before. This was all well and good, and even today is still certainly a valid way to generate an estimate. However, the drawback to this technique is that your estimate is only as good as your expert, and that's assuming he isn't having an off day when he ventures his opinion. For this reason, attempts have been made over the ensuing half-century to develop alternative techniques less dependent upon the subjective judgments of individuals and more on independently verifiable objective criteria and data. The result of those efforts is the spectrum of estimation techniques illustrated in figure 2.1, sequenced in roughly the order they came on the scene. Each technique will be discussed in the following sections.

2.2.1 Expert Opinion

Expert opinion is useful in the absence of quantified, empirical data. It captures the knowledge and experience of practitioners seasoned within a domain of interest, providing estimates based upon a synthesis of the known outcomes of all the past projects to which the expert is privy or in which he participated. But as indicated above, the obvious drawback to this method is that an estimate is only as good as the expert's opinion, and there is no way usually to test that opinion until it is too late to correct the damage if that opinion proves wrong. Also, years of experience do not

endnote #1, *Software Engineering Economics* by Boehm.
necessarily translate into high levels of competency. Moreover, even the most highly competent of individuals will sometimes simply guess wrong.

One estimation method that has been used to capture expert opinion while at the same time trying to mitigate the risk that the judgment of any one expert will be off is called the Delphi technique. Originally applied nearly 50 years ago, Delphi in its most fundamental form is essentially a structured methodology for capturing and blending the judgements of a group of well-informed individuals regarding some issue or question, ultimately with the purpose of driving towards a consensus of opinion regarding the issue at hand.

The origins of the Delphi method are rooted in the early days of the Cold War. In the early 1950s, the RAND Corporation in Santa Monica, CA (a nonprofit research and development institution and the first entity to be called a "think tank"), was running a series of war gaming scenarios, attempting to predict industrial installations within the United States likely to be targeted by a Soviet strategic nuclear attack. As part of these studies, in 1953, two RAND scientists, Olaf Helmer and Norman Dalkey, first developed their technique for statistically aggregating expert opinion for use in forecasting the likelihood of future events. (Hence the name Delphi, after the future-telling Greek oracle of antiquity whose temple was to be found at the village of Delphi on the southern flank of Mt. Parnassos—a name, by the way, apparently provided by others and with which reportedly neither Helmer nor Dalkey was particularly pleased, due to the image it invoked of some mystical being making vague and hard to interpret prognostications, the exact opposite of the...
clearly and mathematical formality they were attempting to bring to the forecasting process.)

The novel contribution of the technique to group decision-making processes is the way that it addresses two contrasting realities of group dynamics: 1) predictions made by persons together are more likely to be correct than predications made by the same persons individually (the so-called "MacGregor Effect," after the researcher who first identified the phenomenon in 1936\(^{15}\)), and 2) useful interchange at group meetings can be hampered by the presence of one or more dominating personalities, as well as with the tendency of group discussions to wander off-topic or get bogged down in minutiae.\(^{16}\)

The Delphi technique in its purest form has the following characteristics:\(^{17}\)

- **Anonymity** - participants in a decision making panel submit their responses to a given question (or questions) independently via survey or questionnaire.

- **Feedback and Iteration** - first round responses to the questionnaire are summarized by the Delphi facilitator using averages, median values, standard deviations, etc., as appropriate; the tabulated results are then returned to the participants, who are asked to revise their responses (again independently) to the questions in light of knowledge of the other participants' original responses.
• Final Consensus - the second round results are again aggregated using statistical measures as appropriate to give the final group answer to the questions posed.

Ideally, over the course of the two rounds, the group will have tended to converge toward a consistent set of responses. The above procedure offers the benefit of the group-derived "MacGregor Effect" while avoiding the drawbacks that can be associated with face-to-face group interaction. The technique is particularly appropriate in the face of little empirical data or a generally accepted theory. The main pitfall to the technique, of course, is the same as that of all expert opinion, namely, the final group judgement can be no better than the totality of the information and expertise available within the group. As noted, however, due to the MacGregor Effect, results still tend to be more reasonable than what might be achievable from experts working on their own.

In the half-century since its first appearance, the technique has gone through much evolution and been adapted for use in a variety of fields, from medicine to business to social policy, so much so that the name Delphi is often applied to methodologies that bear little resemblance to the original technique envisioned by Helmer and Dalkey. (In fact, the moniker has been abused to the point that in the eyes of at least some, the very name Delphi unfortunately has taken on a sinister connotation, invoking Orwellian visions of an "unethical" methodology used to manipulate groups into accepting pre-ordained outcomes.18)
However, when applied as originally intended—as a neutral consensus-taking technique—a variation of the method called Wideband Delphi that allows in-person group interaction between rounds has been found to be very useful in the field of software cost estimation when there is a need to rely on expert judgement.\(^\text{19}\) (Wideband Delphi will be discussed in section 4.2.2 of this report as part of my overall modeling methodology.)

2.2.2 Case Studies & Analogy

Case studies represent an inductive process, whereby estimators and planners try to learn useful general lessons and estimation heuristics by extrapolation from specific examples. They examine in detail elaborate studies describing the environmental conditions and constraints that obtained during the development of previous software projects, the technical and managerial decisions that were made, and the final successes or failures that resulted. They try to root out from these cases the underlying links between cause and effect that can be applied in other contexts. Ideally they look for cases describing projects similar to the project for which they will be attempting to develop estimates, applying the rule of analogy that says similar projects are likely to be subject to similar costs and schedules. The source of case studies can be either internal or external to the estimators' own organizations. "Homegrown" cases are likely to be more useful for the purposes of estimation because they will reflect the specific engineering and business practices likely to be
applied to an organization’s projects in the future, but well-documented case studies from other organizations doing similar kinds of work can also prove very useful.

Shepperd and Schofield did a study comparing the use of analogy against prediction models based upon stepwise regression. They did an analysis on nine data sets (a total of 275 projects), interestingly yielding higher accuracies for estimation by analogy (see section 2.2.4 on statistical regression for possible indications as to why this was the case.) From these experiments they developed a five-step process for estimation by analogy (section 4 of this report discusses a similar but more elaborate stepwise methodology being used as my overall modeling approach):

- identify the data or features to collect
- agree on data definitions and collection mechanisms
- populate the case base
- tune the estimation method
- estimate the effort for a new project

The steps outlined above are easy to list, but each represents a significant effort. Moreover, the early data definition and collection steps will be subject to negotiation and comprise as a balance is sought between what the estimator would ideally like to collect and what is feasible from both an economic and technical standpoint. While every estimator wants his estimate to be as good as possible, a point will be reached beyond which the economic utility of the estimate will not be worth the additional
cost of gathering the information required to ensure the estimate exceeds a given level of fidelity. (This could be said of any estimation technique, not just case studies & analogy.)

The important thing to keep in mind is the strengths and weaknesses of the estimation by case study & analogy approach. The primary strength of the technique is that the estimate is based on experience with actual projects. With careful study, the differences between the new project under consideration and the previous projects can be determined, and thus also the cost adjustments (either up or down) likely to be required for the new project due to those differences. The primary weakness of the technique is that the past projects to which you have access may not be particularly representative of the technical and environmental conditions that will apply during the new project.²¹

2.2.3 Top Down & Bottom Up

Top Down & Bottom Up are complimentary approaches that for thoroughness would ideally be used simultaneously to help generate a project estimate, though this isn't always possible. They can also be used with any of the other estimation techniques discussed in this section as a means of structuring the estimation problem.

With Top Down estimating, an aggregate estimate for the cost of a project in its totality is determined based upon the overall features of the software, using previous similar projects as a starting point (i.e., analogy). Once a total cost is derived, that number is then divided up among the various components of the system to get cost
estimates for the individual system elements. The main advantage of this approach is its ability to capture system level effort such as that associated with component integration and configuration management. It can often also be done fairly quickly. The major drawback is that Top Down estimating can easily miss low level components that nonetheless can be significant cost drivers, either due to technical complexity or sheer number. It also does not offer a detailed breakdown and thus a clearly visible justification for the total overall cost.

Bottom Up estimating takes the reverse approach. It begins with the lowest level components in the system, estimating the cost of each such element individually. The individual component estimates are then summed together to get an estimate for the overall project cost. The advantage of the Bottom Up approach is that the individual component estimates are typically provided by the person (or persons) who will be responsible for doing the work on that component. This means that the estimate is usually based on a very good understanding of the task at hand. The major disadvantage to the method is that it takes longer than and can easily miss the system level costs captured by the Top Down approach. The effect of the latter is that Bottom Up estimates frequently come in too low.22

The way to minimize the risk of not adequately accounting for system level engineering costs is to use the Bottom Up approach in conjunction with a formal work breakdown structure (WBS). Long a standard of engineering practice in the development of both hardware and software, the WBS is a way of organizing project elements into a hierarchy that simplifies the tasks of budget estimation and control. It
helps determine just exactly what costs are being estimated. Moreover, if probabilities are assigned to the costs associated with each individual element of the hierarchy, an overall expected value can be determined from the bottom up for total project development cost.\textsuperscript{23} Expertise comes into play with this method in the determination of the most useful specification of the components within the structure and of those probabilities associated with each component.

A software WBS actually consists of two hierarchies, one representing the software product itself, and the other representing the activities needed to build that product. The product hierarchy (figure 2.2) describes the fundamental structure of the software, showing how the various software components fit into the overall system. The activity hierarchy (figure 2.3) indicates the activities that may be associated with a given software component.

Aside from helping with estimation, the other major use of the WBS is cost accounting and reporting. Each element of the WBS can be assigned its own budget and cost control number, allowing staff to report the amount of time they have spent working on any given project task or component, information that can then be summarized for management budget control purposes.

Finally, if an organization consistently uses a standard WBS for all of its projects, over time it will accrue a very valuable database reflecting its software cost distributions. This data can be used to develop a software cost estimation model tailored to the organization’s own experience and practices.\textsuperscript{24}
2.2.4 Statistical Regression

Statistical regression techniques bring a mathematical formalism to software cost estimation not present in most of the estimation approaches discussed so far. They also feed directly into the parametric modeling techniques that will be discussed in the section following. Their main advantage is that they can remove much of the
subjectivity from the cost estimation process (i.e., two independent estimators applying the same regression methodology on the same data should be able to come up with similar estimates for a given project—not always the case with some of the other techniques). Their main disadvantage is that they usually require a lot of historical project data, and can be easily biased by data that is incomplete, inconsistent or just plain bad.

There are several methods falling under the name "statistical regression," but almost all are variations on the classical approach of general linear regression using least squares. The Ordinary Least Squares (OLS) method is discussed in many books, two good examples being Judge et al\textsuperscript{25} and Weisberg.\textsuperscript{26} The reasons for its popularity include its ease of application\textsuperscript{*} and mathematical simplicity.

A model using the OLS method can be written\textsuperscript{27} as

\[ y_t = \beta_1 + \beta_2 x_{t1} + \ldots + \beta_k x_{tk} + e_t \]  \hspace{1cm} \text{Eq. 2.1}

where

- \( y_t \) = the response variable for the \( t \)th observation.
- \( x_{t1} \ldots x_{tk} \) = predictor (or regressor) variables for the \( t \)th observation.
- \( \beta_1 \) = an intercept parameter.
- \( \beta_2 \ldots \beta_k \) = response coefficients.
- \( e_t \) = a random error variable (usually normally distributed).

\textsuperscript{*} OLS modeling is a standard feature available in commercial statistical modeling software such as Minitab (http://www.minitab.com), S-Plus (http://www.insightful.com), and SPSS (http://www.spss.com).
The OLS method operates by estimating the response coefficients and the intercept parameter by minimizing the least squares error term which is the sum across all observations of $r_i^2$ where $r_i$ is the difference between the observed response and the model predicted response for the $i_{th}$ observation. The result is a line drawn through a set of data that minimizes the difference between the predicted value of an output variable as found on the line and the actual value of that same variable as found in the data, for any given input value found in the data, averaged across all the data available (see figure 2.4).

\[
\text{Min } \sum r_i^2 = \text{Min } \sum (y_{i(a)} - y_{i(p)})^2
\]

where

$y_{i(a)} = y_{\text{actual}}$  \(\bullet\)

$y_{i(p)} = y_{\text{predicted}}$  \(\circ\)

Figure 2.4 - a regression line fitted through empirical data so as to minimize $r_i^2$ across all observations.
This means that all observations have a proportional influence on the model equation. Therefore, if there is an outlier (i.e., an atypical value) in the observations then it will have an undesirable impact on the model, skewing the resulting line in a direction that increases the error between the predicted and actual values of the output variable that are most typically found in order to reduce the error between the predicted and actual values of that single atypical observation. The result is that while trying to compensate for that unusual case, the model will have weakened its ability to provide a good estimate under the conditions that are most usually encountered (see figure 2.5, where the thick dashed line indicates the regression line in figure 2.4 before the value of $y_i(\text{actual})$ was shifted—note that this new model in most cases now tends to overestimate more frequently and by a much wider margin than it underestimates whereas the original model tended to over and underestimate with about the same frequency and by about the same margin).

The conditions under which the OLS method is appropriate to use (along with comments about how these conditions are nonetheless frequently violated in the development and application of statistically-based software cost estimation models) are listed below. The OLS method is appropriate when

- a lot of data are available. This indicates that there are many degrees of freedom available and the number of observations is many more than the number of variables to be predicted. (Collecting data is one of the biggest challenges in the software field for several reasons, including lack of
commitment by higher management, co-existence of several development processes, lack of proper interpretation of the process, and budget concerns to name just a few.)

\[ \text{Min } \sum r_i^2 = \text{Min } \sum [y_i(a) - y_i(p)]^2 \]

where
\[ y_i(a) = y_i(\text{actual}) \]
\[ y_i(p) = y_i(\text{predicted}) \]

Figure 2.5 - a regression line skewed by an extreme outlier.
• no data items are missing. (Software data with missing information is often reported when there is limited time and budget for the data collection activity, or when there is a lack of understanding of the data being requested.)

• there are no outliers. (Extreme cases are very often reported in software engineering data due to misunderstandings or lack of precision in the data collection process, or due to the use of different software development processes.)

• the predictor variables are not correlated. (Most of the existing parametric software cost estimation models have parameters that are at least somewhat correlated to each other. Eliminating these correlations in software data to the level that would be desirable is often very difficult.)

• the predictor variables have an easy interpretation when used in the model. (This is very difficult to achieve because it is not easy to make valid assumptions about the form of the functional relationships between predictors and their probability distributions.)

• the predictor variables are either all continuous (e.g., database size) or all discrete (e.g., the software development process is either waterfall or spiral).
Parametric software cost models typically must include both continuous and discrete input parameters if they are to reflect the real-world factors that are known to impact software development costs.

The main difficulty in using regression techniques for software cost estimation is that software data is notorious for being noisy, correlated and incomplete, which violates at least three of the conditions noted above. As such any software data used for statistically-based modeling must undergo a certain amount of "conditioning," which will be described in section 4.2.3 while discussing my overall research approach.

2.2.5 Parametric Modeling

Parametric software cost estimation models generate estimates based on mathematical relationships between dependent variables (typically effort and/or schedule) and one or more independent variables or parameters (e.g., size of the job, job complexity, personnel capabilities, design characteristics, etc.). The input parameters will normally characterize the nature of the software job to be done, plus the environmental conditions under which the work will be performed and delivered. As an output parameter, effort is usually substituted as a proxy for cost,* allowing the

* There are two reasons for this: 1) financial data is usually held very close to the vest by corporations, and even with the guarantee of a non-disclosure agreement between company and independent data collector, businesses are generally unwilling to release actual cost information to outside modelers; 2) effort is a more fundamental quantity for interpretation that can be more readily normalized across diverse organizations—it also does not need to take into account the time value of
individual estimator to apply whatever currency-unit per person-month conversion factor is appropriate in his or her situation.

The definition of the mathematical relationships noted above between the input and output parameters is the heart of parametric modeling. These relationships are usually based upon statistical analyses of large amounts of software engineering development data. Regression models as described in the previous section are used to tease out appropriate numeric values for constants and finite, discrete-valued variables that capture mathematically the design and development conditions as described by the input parameters and relate these in linear or nonlinear equations to the output parameters. Using statistical regression and other mathematical techniques, the constants, variables, and forms of the equations are ideally all optimized to minimize the difference between cost estimates derived from the parametric model and what eventually winds up being the true cost of developing given software systems.

Parametric software cost models typically can be applied at either the overall system level, or at any lower software component level that can be bounded or described as a distinct work unit or design element. Some parametric models also allow their estimated overall costs to be partitioned temporally among the various phases of the software development life cycle (i.e., requirements definition, design, money (though periodic model recalibrations become necessary as technological innovations over time improve productivity)).
programming, test, etc.—again, see the project activities listed in figure 1.1). This ability aids in the planning for necessary project personnel loading levels over time.

The advantages of parametric estimation models once validated are that they are fast and easy to use, they (generally) require little detailed information, and—if designed properly—they can capture total system or component-level costs, including costs related to integration activities. Moreover, in addition to the speed and ease with which they can be applied, well-designed parametric models can be as accurate as other estimating techniques, if not more so.29

The major disadvantages of these kinds of models is that they are difficult and time consuming to develop, and they suffer all the drawbacks of any statistically-based models in that they require a lot of clean, complete, uncorrelated data to be properly validated.

Despite these problems, their utility in terms of their potential speed and accuracy is so great, many such parametric software cost estimation models have been developed over the years, COCOMO itself being among the earliest and most widely accepted of these models. Being an extension of COCOMO II, COCOTS is also being developed as a parametric model.

To illustrate the utility of these kinds of models, figure 2.6 presents a version of a classic Norden-Rayleigh curve distribution of engineering project labor over time.30 31 (This model has provided the foundations for several subsequent parametric estimation models, including SLIM,32 SEER-SEM (see section 2.3.5), and the
estimated labor loading by project phase of COCOMO II. This makes it a useful example for discussion here.)

The Norden-Rayleigh model suggests that the number of people hired onto a software project at any given time during its development cycle follows a somewhat modified bell-shaped distribution with a long tail to the right, indicating that workers tend to join a project more rapidly in its early stages than they leave the project in its latter stages. In the curve shown in figure 2.6, if $t_D$ represents the point in time at which the project is expected to attain its peak labor loading, then the fraction of that
peak personnel loading that nominally could be expected to be needed on the project at any given time $t$ can be estimated by

$$\% \text{ Max PPL} = \left( \frac{t}{t_D} \right) e^{\frac{(t_D^2 - t^2)}{2t_D^2}}$$

Eq. 2.2

where

$\% \text{ Max PPL} =$ percentage of the maximum project personnel loading.

$t_D =$ the point in time at which the maximum project personnel loading is reached.*

$t =$ time.

The specific shape of the curve in figure 2.6 can be affected by environmental conditions (e.g., project difficulty, project domain) represented by parameters not indicated in equation 2.2. However, Putnam has shown that in general, Rayleigh-based models can provide good approximations to the labor distribution on software projects.33

Descriptions of several more of the more prominent parametric software cost models that address traditional software development methods (Putnam's SLIM model, the Jensen Model, Checkpoint, etc.) are available in Boehm et al34 and the DoD Parametric Handbook.35 Five existing parametric models that specifically address the use of COTS software components are discussed in section 2.3 of this dissertation. Finally, an overview of COCOMO II more detailed than that given in
section 1.1 is provided in section 2.4. This will help to further illuminate the relationship between COCOMO and COCOTS.

2.2.6 Neural Networks

According to Gray and McDonell, neural networks is the most common software estimation modeling technique used as an alternative to Ordinary Least Squares regression (I believe they class parametric models among OLS regression techniques). Neural Nets are estimation models that can be “trained” using historical data to produce ever better results by automatically adjusting their algorithmic parameter values to reduce the difference between known actual cost numbers and model predictions. Gray and McDonell go on to describe the most common form of a neural network used in the context of software estimation, a “back-propagation trained feed-forward” network (see figure 2.7).

* As a practical matter, \( t_d \) is often assumed to occur at the midpoint of a project development schedule if that schedule is determined by means that relate estimated development schedule to estimated total project development effort.
Figure 2.7- a neural network estimation model.

The development of such a neural model is begun by first developing an appropriate layout of "neurons," or connections between network nodes. This includes defining the number of layers of neurons, the number of neurons within each layer, and the manner in which they are all linked. The weighted estimating functions between the nodes and the specific training algorithm to be used must also be determined. Once the network has been built, the model must be trained by providing it with a set of historical project data inputs and the corresponding known actual values for project schedule and/or cost. The model then iterates on its training algorithm, automatically adjusting the parameters of its estimation functions until the model estimate and the actual values are within some pre-specified delta. The
specification of a delta value is important. Without it, a model could theoretically become overtrained to the known historical data, adjusting its estimation algorithms until it is very good at predicting results for the training data set, but weakening the applicability of those estimation algorithms to a broader set of more general data.

Wittig\textsuperscript{37} has reported accuracies of within 10\% for a model of this type when used to estimate software development effort, but caution must be exercised when using these models as they are often subject to the same kinds of statistical problems with the training data as are the standard regression techniques used to calibrate more traditional models. In particular, extremely large data sets are needed to accurately train neural networks with intermediate structures of any complexity. Also, for negotiation and sensitivity analysis, the neural networks provide little intuitive support for understanding the sensitivity relationships between cost driver parameters and model results. They encounter similar difficulties for use in planning and control.

2.2.7 Dynamic Simulation

Dynamic simulation techniques explicitly acknowledge that software project effort or cost factors change over the duration of the system development; that is, they are dynamic rather than static over time. This is a significant departure from the other techniques highlighted in this report, which tend to rely on static models and predictions based upon snapshots of a development situation at a particular moment in time. However, factors like deadlines, staffing levels, design requirements,
training needs, budget, etc., all fluctuate over the course of development and cause corresponding fluctuations in the productivity of project personnel. This in turn has consequences for the likelihood of a project coming in on schedule and within budget—usually negative. The most prominent dynamic techniques are based upon the system dynamics approach to modeling originated by Jay Forrester nearly forty years ago.38

System dynamics is a continuous simulation modeling methodology whereby model results and behavior are displayed as graphs of information that change over time. Models are represented as networks modified with positive and negative feedback loops. Elements within the models are expressed as dynamically changing levels or accumulations (the nodes), rates or flows between the levels (the lines connecting the nodes), and information relative to the system that changes over time and dynamically affects the flow rates between the levels (the feedback loops).

Figure 2.8 taken from Madachy39 shows an example of a system dynamics model demonstrating the famous (in software circles) Brooks' Law,40 which states that "adding manpower to a late software project makes it later." Brooks' rationale is that not only does effort have to be reallocated to train the new people, but the corresponding increase in communication and coordination overhead grows exponentially as people are added.

The dynamic model as shown in the figure illustrates Brooks' concept based on the following assumptions:
• New people need to be trained by experienced people to improve their productivity.

• Increasing staff on a project increases the coordination and communication overhead.

• People who have been working on a project for a while are more productive than newly added people.

Figure 2.8 - Madachy's system dynamics model of Brooks' Law.
As can be seen in figure 2.8, the model shows two flow chains representing software development and personnel. The software chain (seen at the top of the figure) begins with a level of requirements that need to be converted into an accumulation of developed software. The rate at which this happens depends on the number of trained personnel working on the project. The number of trained personnel in turn is a function of the personnel flow chain (seen at the bottom of the figure). New people are assigned to the project according to the personnel allocation rate, and then converted to experienced personnel according to the assimilation rate. The other items shown in the figure (nominal productivity, communication overhead, experienced personnel needed for training, and training overhead) are examples of auxiliary variables that also affect the software development rate.

Mathematically, system dynamics simulation models are represented by a set of first-order differential equations.\(^{41}\)

\[
x'(t) = f(x, p)
\]

Eq. 2.3

where

\[
x'(t) = \text{the incremental rate of change of the levels (states) in the model at time } t.
\]

\[
x = \text{a vector describing the levels (states) in the model.}
\]

\[
p = \text{a set of model parameters.}
\]
\[ f = \text{a nonlinear vector function dependent on time } t. \]

\[ t = \text{time.} \]

Within the last ten years this technique has been applied successfully in the context of software engineering estimation models. Abdel-Hamid and Madnick have built models that will predict changes in project cost, staffing needs and schedules over time, as long as the initial proper values of project development are available to the estimator. They have also applied the technique in the context of software reuse, demonstrating an interesting result. They found that there is an initial beneficial relationship between the reuse of software components and project personnel productivity, since less effort is being spent developing new code. However, over time this benefit diminishes if older reuse components are retired and no replacement components have been written, thus forcing the abandonment of the reuse strategy until enough new reusable components have been created, or unless they can be acquired from an outside source (e.g., a COTS product vendor).

More recently, Madachy used system dynamics to model an inspection-based software life cycle process. He was able to show that performing software inspections during development slightly increases programming effort, but decreases later effort and schedule during testing and integration. Whether there is an overall savings in project effort resulting from that trade-off is a function of development phase software error injection rates, the level of effort required to fix errors found during testing, and the efficiency of the inspection process. For typical industrial
values of these parameters, the savings due to inspections considerably outweigh the costs.

   Dynamics-based techniques are particularly good for planning and control, but are also particularly difficult to calibrate.

2.2.8 Bayesian Analysis

   Named after the 18th century English mathematician and clergyman the Reverend Thomas Bayes who first proposed the technique in the 1760s,48 Bayesian analysis is a mode of inductive reasoning that takes into account our ability to learn new information about a given matter or event of concern. In particular, it offers ways of quantifying our subjective "degree of belief" about some assertion or event, given what we have observed regarding some related assertion or subsequent event.

   More specifically, Reverend Bayes developed a formula for determining the probability that some prior event occurred or unobserved state of nature exists, given that a subsequent event or other state is definitively known to have occurred or been positively observed. It is important to understand, however, that the "probability" being discussed here is a characterization of one's subjective belief in the certainty of something rather than an objective measure of the frequency with which that something is likely to occur or be true.49 (This latter interpretation is the classical definition of "probability" underlying traditional statistical theory.)

   To use Bayesian analysis, you must be able to assign a subjective probability density to the event or state under question that reflects your belief in the certainty
that the event has occurred or the state of nature exists, prior to sampling any data
that might offer more clues about the situation. If we call that event or state \( \beta \), then
the probability density function \( f(\beta) \) that reflects your level of certainty about the
occurrence or value of \( \beta \) before any additional information about \( \beta \) is collected, is
called the unconditional prior distribution of \( \beta \). The more certain you are about the
true value of \( \beta \) prior to any sampling, then \( f(\beta) \) should be chosen with a
proportionally smaller variance. The converse is also true; the less certain you are
about \( \beta \), then an \( f(\beta) \) with a wider variance should be selected.

After additional information has been gathered, your belief in the certainty of \( \beta \)
may change. If we call that new information \( Y \), then the probability density function \( f
(\beta|Y) \) that reflects your new level of certainty about \( \beta \) given that has \( Y \) has been
observed is called the conditional posterior distribution of \( \beta \). In other words, this is
the new conditional probability distribution of \( \beta|Y \), reflecting your changed belief in
the likelihood that \( \beta \) is true or has occurred given that you know for a certainty that \( Y \)
is true or has occurred.

The probability density function \( f(Y|\beta) \) is the probability that \( Y \) will be observed
or is true given that \( \beta \) is known to have occurred or be true.

The probability density function \( f(Y) \) is the probability that \( Y \) will be observed or
is true regardless of whether \( \beta \) is true or false; that is, \( f(Y) \) is the probability that \( Y \) is
true given that \( \beta \) is true summed with the probability that \( Y \) is true given that \( \beta \) is
false.
Case 1: variance around estimate from experts smaller than that around value from empirical sample

Figure 2.9 - influence of variance on a given estimated value determined by Bayesian statistical methods.

Case 2: variance around estimate from experts greater than that around value from empirical sample
The conditional posterior density of $\beta$ is then calculated as follows:

$$f (\beta | Y ) = \frac{f (Y | \beta ) f (\beta )}{f (Y )} \quad \text{Eq. 2.4}$$

where

$f (\beta | Y ) = \text{the conditional probability of } \beta \text{ given } Y.$
$f (Y | \beta ) = \text{the conditional probability of } Y \text{ given } \beta.$
$f (\beta ) = \text{the probability of } \beta \text{ independent of } Y.$
$f (Y ) = \text{the probability of } Y \text{ independent of } \beta.$

The above is known as Bayes' theorem and says that the probability of $\beta$ having occurred given that $Y$ is known to have occurred, is equal to the probability of $Y$ having occurred given that $\beta$ is known to have occurred multiplied by the prior unconditional probability of $\beta$ having occurred before $Y$ is known, all divided by the probability that $Y$ will occur regardless of whether $\beta$ ever occurred.$^{50}$

In terms of how Bayesian analysis can be applied to software cost estimation, it allows empirically sampled cost estimation data to be combined with intuitive expert opinion and other informally gathered information in a logically consistent manner useful for making inferences. In this context, estimates based solely on expert opinion represent the prior distribution while revised estimates incorporating empirical information with that expert judgement represent the conditional posterior distribution. The combined revised estimates are produced using Bayes' theorem to
create a "post-data" or posterior distribution for estimates. The posterior distribution is determined by the variances of the prior and sample or empirical information. If the variance of the estimates produced by prior expert judgement is smaller than the variance of the estimates derived from empirically sampled data, then a higher weight is assigned to the expert opinion (Case 1 in figure 2.9). On the other hand, if the variance of the sample information is smaller than the variance of the prior information, then a higher weight is assigned to the sample data causing the posterior estimate to be influenced more by the empirical data (Case 2 in figure 2.9).51

(Typically, what happens during the course of developing an estimation model using Bayesian techniques is that in the beginning, you find yourself in a situation akin to Case 1. This is because at this stage expert opinion is usually all you have to go on. But ideally, over time, as your model building efforts continue, you gather sufficient empirical information for calibration purposes to swing your model from a predominately opinion-based construct to one anchored firmly on an empirical footing as illustrated in Case 2.)

Bayesian analysis has all the advantages of classical linear regression based on observed empirical data plus the ability to include the subjective prior knowledge of experts. It attempts to balance the risks associated with imperfect data gathering against those risks associated with relying exclusively on expert judgement. As much as hard empirical data is prized, the reality is that empirically-based software engineering data is typically scarce and incomplete. However, much good subjective information based on years of personal experience with software processes and the
study of the factors that most affect software development effort, cost, schedule, and product quality is often available from experts within a software development organization. But data of this kind doesn’t easily lend itself to analysis with classical statistical techniques. Bayesian techniques allow this information to be used along with collected sample data in the cost estimation process.

2.3 Existing Parametric COTS-related Cost Estimation Models

One of the key assertions justifying the research being discussed in this report is that empirically-based cost estimation models that address CBS design are few in number. This next section will discuss five such models known to exist and which have helped to inform the development of COCOTS. (The seeming brevity of some of these discussions simply reflects how little information has been openly published about these models, sometimes because of proprietary concerns, and sometimes because no work beyond what is described herein was apparently ever pursued.)

2.3.1 Stutzke Model

Of the few CBS models out there, an interesting one has been proposed by Richard Stutzke of SAIC. It is still bare bones, but it is centered on the issue of COTS volatility, that is, the frequency with which a COTS vendor releases new versions of its software, and the extent of the changes the vendor makes in the product with each new release. His model suggests a way of quantifying the added cost associated with using a COTS product that has significant volatility.
Stutzke proposes the following formula:

\[ ECV = CV \cdot AC \cdot IS \cdot (CS + CC) \]  

Eq. 2.5

where

- **ECV** = extra CBS cost due to volatility of the given COTS component.
- **CV** = component volatility (number of new releases of the COTS component over life of the project).
- **AC** = architectural coupling (number of other components which interface with the given COTS component).
- **IS** = apparent interface size in terms of the number of entry points, procedures, functions or other methods used to access the COTS component, weighted by the number of arguments passed.
- **CS** = cost of screening the COTS component and all the other components with which it interfaces to determine the impact of a new release.
- **CC** = cost of making changes to impacted components.

As of this writing no attempt has yet been made to implement this model. It also addresses only one aspect associated with integrating COTS software, the issue of COTS component volatility, but it is an important aspect.

### 2.3.2 Alternate SAIC Model

Still another approach to modeling COTS integration costs has been taken at SAIC. This second model addresses a few of the development costs, but focuses more on the end user costs of adopting a particular COTS software component:

\[ COTS IC = (LC \cdot NL) + TC + GC \]

Eq. 2.6

where
COTS IC = COTS product integration cost.

LC = cost of COTS product license.

NL = number of COTS product licenses required.

TC = COTS product training cost.

GC = COTS interface or glue code cost.

Again, this model highlights some important sources of cost associated with the use of COTS software, but ignores the details of determining the last term, the cost of developing the glue code needed to integrate the COTS product into the larger system being built. It also ignores the cost of assessing and selecting which COTS products to use.

2.3.3 Ellis Model

A CBS model that attempts to address the issue of estimating the cost of developing COTS interface or glue code has been described by Tim Ellis \(^{54}\) of what was then Loral Federal Systems (now Lockheed Martin Federal Systems, and before Loral was IBM Federal Systems). This model was implemented and calibrated to a number of internal Loral COTS integration projects, but active development of the model apparently stopped after a few years. \(^*\) However, as of May 1995, an accuracy of (+/-) 15% was being claimed for its effort predictions against the database of internal Loral projects.

\(^*\) Personal conversation between myself and a former Loral employee while I was giving a COTS modeling seminar at the offices of the Software Productivity Consortium in Herndon, VA, April 2000.
The details are proprietary, but Ellis describes the model in these general terms:

\[
\text{COTS GC} = \text{WU} \cdot \text{PR} \tag{2.7a}
\]

where

\[
\text{WU} = f(\text{Size, Drivers}) \tag{2.7b}
\]

\[
\text{PR} = \frac{\text{EF}}{\text{WU}} \tag{2.7c}
\]

and

\[
\text{COTS GC} = \text{effort to create interface or glue code for a COTS product.}
\]

\[
\text{WU} = \text{internally defined Loral "work units."}
\]

\[
\text{Size} = \text{the size of the COTS interface or glue code in function points.}^* \tag{55}
\]

\[
\text{Drivers} = \text{a set of seventeen COTS integration cost drivers.}
\]

\[
f = \text{a Loral proprietary function that relates the amount of glue to be written and the ratings of individual glue code cost drivers to a given number of work units.}
\]

\[
\text{PR} = \text{productivity in terms of the effort required to produce one work unit.}
\]

\[
\text{EF} = \text{effort in labor-months.}
\]

Of the five CBS-oriented cost models that are being discussed in this section, the Ellis model has definitely had the greatest influence on the current form of COCOTS. As detailed in my preliminary COTS modeling study (again, see chapter 1 endote #18), the seventeen cost drivers appearing in the Ellis model served as one of

---

*Function point counting is a technique for sizing a given software development job alternatively to the traditional sizing technique of counting source lines of code. It tries via a set of objectively defined measures to quantify directly the functionality being delivered by the software as opposed to
the starting points in the definition of the cost drivers for the initial COTS model resulting from that first effort. As such, the Ellis drivers are presented here:

- **Product Maturity**: Measures the length of time the product has been in the marketplace, existence of extensive alpha/beta testing programs, size of the market segment, number of bug fixes per release, and adherence to industry standards.

- **Vendor Maturity**: Measures the length of time the vendor has been in the business, vendor reputation, and size of product line.

- **Configurability/Customization**: Number of configuration options, and effort needed to customize the product.

- **Installation Ease**: Effort needed for product installation.

- **Ease to Upgrade**: Measures the level of difficulty to upgrade the COTS software from one release to the next and the impact to the applications being developed.

- **Vendor Cooperation**: Represents willingness of vendors to modify their product based on suggestions or enhancements recommended by the user. The more cooperative the vendor, the more functionality is provided thus reducing new development and glue code.

---

quantifying the number of lines of code in the software required to provide that functionality. See chapter endnote #55 for a detailed discussion of function points and their application.
• **Product Support Services:** Types of services offered by the vendor to support the product (e.g., 24-hour hotline, seminars, trouble ticketing, etc.).

• **Product Support Quality:** Responsiveness of the vendor to user questions.

• **User, Administrator, & Installation Documentation:** Quality of documentation offered.

• **Ease of Use for End User:** How intuitive is the product for the end user?

• **Ease of Use for Administrator:** How intuitive is the product for the administrator?

• **End User & Administrative Training:** Types and quality of training available.

• **Administrative Effort:** Amount of time spent by system administrator to regularly maintain the system.

• **Portability:** Portability of the product between platforms.

• **Previous Product Experience:** Amount of experience that personnel have had developing/using/integrating the product.

• **Expected Release Frequency:** Amount of time between product upgrades and releases. For every product upgrade, testing must be performed to ensure that no new incompatibilities have been introduced. This is a key cost driver that can adversely affect the integration phases of COTS products.

• **Application or System COTS Package:** Is the COTS software an application or is it a system level/infrastructure type of product?
2.3.4 PRICE-S

An antecedent of the PRICE-S cost model ("S" for software) was originally developed at RCA Corp. for use internally on software projects during the era of the Apollo moon program. However, motivated by the success of their commercially offered parametric engineering hardware cost model PRICE-H ("H" for hardware), RCA decided in the mid-1970s to develop their internal software cost model into a related commercial product. PRICE Systems was thus born in 1975 as an RCA subsidiary and now—after having gone through a series of acquisitions during the aerospace industry consolidations of the 1990s—is an independent, privately held company, offering a variety of related estimation models for both hardware and software.56 Their software model, PRICE-S, developed from that initial internal RCA software model, was first offered to the general public in 1977. Since then, it has been used for estimations on many US Government software projects, including several within the DoD and in particular on projects within NASA.

Being proprietary, the deep details of the model's equations have never been released in the public domain, although an overall description of the model's central algorithms was published in 1988.57 In general, PRICE-S consists of three submodels that enable estimating costs and schedules addressing a variety of software development activities:

The Acquisition submodel forecasts software development costs and schedules. It is calibrated to most of the major software domains, including business systems, communications, command and control, avionics, and space systems. It addresses
current software development practices such as reengineering, automated code
generation, spiral development, rapid development, rapid prototyping, object-
oriented development, as well as supporting software productivity measurement
activities.

The Sizing submodel estimates the size or amount of software to be developed.
Sizing can be done in source lines of code, function points or Predictive Object
Points.*

The Life Cycle Cost submodel is used for rapid and early costing of the
maintenance and support phase of a large software project. It is used in conjunction
with the Acquisition submodel, which provides the development costs and original
design parameters.

The current version of PRICE-S addresses COTS modeling in the Acquisition
and Life Cycle submodels as up-front and continuing line item cost elements,
respectively, though the details of that modeling remain proprietary.

2.3.5 SEER-SEM

SEER-SEM (standing for System Evaluation and Estimation of Resources -
Software Estimation Model) is a product offered by Galorath, Inc.* (which like
PRICE Systems offers a suite of related hardware and software estimation tools) and
is based on a model originally published by Randall Jensen* (which in turn is related

---

* Like function points, Predictive Object Point counting is a more recent alternative to sizing software
in source lines of code. It is a technique intended for sizing software being built using newer object
through Putnam's SLIM model to the Norden-Rayleigh curves discussed in section 2.2.5). On the market some 15 years, during that time SEER-SEM has evolved into a sophisticated tool supporting top-down and bottom-up estimation methodologies. Its modeling equations are proprietary, but take a parametric approach to estimation. The scope of the model is wide. It covers all phases of the project life cycle, from early specification through design, development, delivery and maintenance. It handles a variety of software system configurations, from stand-alone to client-server to distributed. It models the most widely used development methods and languages. Development modes covered include object oriented, reuse, COTS, spiral, waterfall, prototype and incremental development. Languages covered are third and fourth generation languages (C++, FORTRAN, COBOL, Ada, etc.), as well as application generators. * It allows staff capability, required design and process standards, and levels of acceptable development risk to be input as constraints.

* Application Generators (AG) are high level programming tools that allow developers to build software applications by specifying the needed software functionality for which the AG will then automatically generate the appropriate computer source code.
Figure 2.10 is adapted from a Galorath illustration and shows gross categories of model inputs and outputs, but each of these represents dozens of specific input and output possibilities and parameters. The reports available from the model—covering all aspects of input and output summaries and analyses—number in the hundreds.

Features of SEER-SEM include the following:

- Allows probability level of estimates, staffing and schedule constraints to be input as independent variables.
- Facilitates extensive sensitivity and trade-off analyses on model input parameters.
• Organizes project elements into work breakdown structures for convenient planning and control.

• Displays project cost drivers.

• Allows the interactive scheduling of project elements on Gantt charts.

• Builds estimates upon a sizable knowledge base of existing projects.

Model specifications include these:

Parameters: size, personnel, complexity, environment and constraints (each with many individual parameters); knowledge base categories for platform and application, development and acquisition method, applicable standards, plus a user-customizable knowledge base.

Predictions: effort, schedule, staffing, defects and cost estimates; estimates can be schedule or effort driven; constraints can be specified on schedule and staffing.

Risk Analysis: sensitivity analysis available on all smallest/most likely/largest values of output parameters; probability settings for individual WBS elements adjustable, allowing for sorting of estimates by degree of WBS element criticality.

Sizing Methods: function points, both IFPUG* sanctioned plus an augmented set; lines of code, both new and existing.

---

* International Function Point Users Group, which is the recognized governing body maintaining the standards and practices of function point metrics.
Outputs and Interfaces: many capability metrics, plus hundreds of reports and charts; trade-off analyses with side-by-side comparison of alternatives; integration with other Microsoft™ Windows applications plus user customizable interfaces.

Again, as with the PRICE-S model, the details of the COTS related cost estimation algorithms within SEER-SEM remain proprietary, but this model also appears to handle COTS-related costs as line item elements.

2.4 Description of COCOMO II

An overview of COCOMO II was first presented in section 1.1. Of the three COCOMO II submodels discussed there (Application Composition, Early Design, Post-Architecture), it is the Post-Architecture model which is the most detailed. It is also the model with the most direct connection to COCOTS. As such, it is this model that is elaborated upon here.

2.4.1 Effort Estimation

To begin, COCOMO II builds upon the basic modeling equation shown below as equation 2.8. This is a standard form followed by many of the parametric software estimation models mentioned previously.

\[ \text{Effort} = A \cdot (\text{Size})^B \]  

Eq. 2.8

where

Effort = software development effort (usually given in person-months).
Size = size of the software system under development (typically indicated in source lines of code *64 65 but other measures are sometimes used as has been indicated previously—e.g., function points, object points, etc.).

A = a multiplicative conversion constant relating software program size to development effort.

B = an exponential factor that accounts for nonlinear economies or diseconomies of scale that may accrue as software increases in size. (As a rule, software tends to exhibit diseconomies of scale due to the exponentially increasing number of interfaces that must be managed as components are added to the system, as well as the increased overhead that goes along with managing more workers and the communication between them.65)

* Source lines of code (SLOC) has been the traditional method for sizing software projects for nearly a half-century. The popularity of this metric stems from the fact that a line of code is an easy concept to grasp—more often than not it is the thing which is actually being produced by a programmer as he does his work; automated tools exist that can count the exact number of lines of a code in a software product after it is completed; and other sizing methods (e.g., function points) are difficult to learn to apply properly. However, even given all these positives, sizing in SLOC has drawbacks, foremost of which is that deciding just what exactly constitutes "a line of code" is not always so straightforward. In the days when computer programs were still created using 80 character paper punch cards, you typically just counted the number of cards in your completed stack, sometimes excluding comment cards from your final count. These days, however, code is written using real-time editors on a monitor screen, using structured formats (see chapter endnote #64 for a discussion of structured software design) that can cause the same line of executable code written by two different people to extend and wrap across the screen in completely different ways depending on the formatting scheme each programmer uses. The choice of computer language itself (e.g., FORTRAN vs. C++ vs. LISP) also can make a huge difference in the definition of "a line of code." For these reasons, COCOMO uses a SEI-developed standard (see chapter endnote #65) defining SLOC in terms of executable, non-comment, logical lines of code as opposed to commented, physical SLOC, to compensate for lack of uniformity in program formatting practice.

Another drawback to estimating software size in SLOC is that to do it reasonably accurately, you need to know a lot of detail regarding the final structure of the program, down to the lowest levels. This makes SLOC sizing problematic to apply in early proposal stages of most software projects.

And finally, SLOC counts these days often simply don't apply, as in the example given earlier of software based on object oriented design (see footnote page 68).

Even with all these drawbacks, however, sizing in SLOC is still the most commonly used software sizing method.
The Post-Architecture model of COCOMO II refines the basic equation shown above as follows:

\[
\text{Effort}_{\text{COC}} = A_{\text{COC}} \left( E_{\text{size}_{\text{COC}}} \right)^{E_{\text{COC}}} \prod_{i=1}^{17} \text{EM}_{\text{COC}(i)} 
\]

Eq. 2.9a

where

\[
E_{\text{size}_{\text{COC}}} = \text{Size}_{\text{COC}} \cdot (1 + \text{REVL}/100) 
\]

Eq. 2.9b

and

\[
E_{\text{COC}} = B_{\text{COC}} + 0.01 \sum_{j=1}^{5} \text{SF}_{\text{COC}(j)} 
\]

Eq. 2.9c

and

\[
\text{Effort}_{\text{COC}} = \text{software development effort (usually given in person-months).}
\]

\[
E_{\text{size}_{\text{COC}}} = \text{effective size of the software system under development after adjusting for rework that must be done as a result of changes in requirements.}
\]

\[
\text{Size}_{\text{COC}} = \text{absolute size of the software system under development (either in source lines of code or function points).}
\]

\[
\text{REVL} = \text{estimated percentage of code that must be reworked during development due to changes or evolution in requirements or COTS component volatility but explicitly not as a result of programmer error.}
\]

\[
A_{\text{COC}} = \text{a multiplicative conversion constant relating software program size to development effort, now representing the productivity that typically obtains when project conditions allow all seventeen linear "effort multiplier" parameters EM}_{\text{COC}(i)} \text{ in the model to be assigned their baseline "nominal" ratings, thus reducing their collective impact to nil.}
\]

\[
\text{EM}_{\text{COC}(i)} = \text{"effort multipliers" that either increase or decrease the nominal effort estimate given by the equation based upon characterizations of the environmental conditions that exist while the system is under development; their nominal value is 1.0.}
\]
\[ E_{\text{COC}} = \text{an exponential factor that accounts for nonlinear economies or diseconomies of scale that may accrue as software increases in size and which in turn is now a function of a constant } B_{\text{COC}} \text{ and five "scale factors" } S_{F_{\text{COC}(j)}}. \]

\[ B_{\text{COC}} = \text{a constant appearing in the exponential term that represents the costs or savings that still obtain even when project conditions allow the absolute best possible ratings to be assigned each of the scale factors } S_{F_{\text{COC}(j)}}, \text{ reducing their collective impact to nil; the 2000 calibration of COCOMO II currently assigns a value of 0.91 to } B_{\text{COC}}, \text{ which implies that under the best possible system-wide conditions, } \text{economies of scale become evident as software increases in size, which is the inverse of what more typically has proven to be the case.} \]

\[ S_{F_{\text{COC}(j)}} = \text{"scale factors" characterizing project conditions that have been shown to have nonlinear impacts on software development effort determining whether economies or diseconomies of scale will likely present during the development.} \]

In broad terms, the seventeen effort multipliers \( E_{M_{\text{COC}(i)}} \) new to this equation address a) characteristics of the software product itself; b) the virtual platform (meaning both the hardware and the infrastructure software upon which the system is being developed and ultimately expected to perform); c) the personnel who are developing the software system; and d) project development conditions in terms of the use of automated tools to aid in the software development, co-location (or lack thereof) of the development team, and any requested acceleration in the development schedule.

The five scale factors \( S_{F_{\text{COC}(j)}} \) impacting economies or diseconomies of scale resulting from a given project size address a) whether or not a software project is similar to projects performed by the developing organization in the past; b) how rigidly the final software product must adhere to the originally specified project
requirements; c) how many of the known significant risks to a successful project outcome have been addressed by the choice of system architecture and design; d) how well all the stakeholders to the project (users, developers, funders, procurers, etc.) work together and share common objectives; and e) the maturity of the development processes that will be applied during the life of the software project.

The basic form of the COCOMO II Post-Architecture model, including the effort multipliers and scale factors, have influenced the current form of parts of COCOTS. As such, like the Ellis model drivers shown in section 2.3.3, the definitions for COCOMO II Post-Architecture scale factors and effort multipliers are presented here:

Exponential scale factors $SF_{\text{COCO}}$:

- **Precededness (PREC):** If the product is similar to several that have been developed before then the precededness is high.

- **Development Flexibility (FLEX):** Captures the amount of constraints the product has to meet. The more flexible the requirements, schedules, interfaces, etc., the higher the rating.

- **Architecture/Risk Resolution (RESL):** Captures the thoroughness of definition and freedom from risk of the software architecture used for the product.

- **Team Cohesion (TEAM):** Accounts for the sources of project turbulence and extra effort due to difficulties in synchronizing the project’s stakeholders: users, customers, developers, maintainers, interfacers, others.
• **Process Maturity (PMAT):** Based upon the SEI’s Capability Maturity Model (CMM) ratings of organization-wide software development process maturity.* 67

Effort multipliers $EM_{OD(i)}$:

*Product Drivers*

• **Required Software Reliability (RELY):** Measure of the extent to which the software must perform its intended function over a period of time.

• **Database Size (DATA):** Measure of the affect large data requirements has on product development.

• **Product Complexity (CPLX):** Measures complexity of software under development in five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations.

• **Required Reusability (RUSE):** Accounts for the additional effort needed to construct components intended for reuse on the current or future projects.

• **Documentation Match to Life Cycle Needs (DOCU):** Measures the suitability of the project’s documentation to its life cycle needs.

---

*The SEI Capability Maturity Model (CMM) describes a framework for characterizing an organization’s processes for developing and maintaining software via a five level classification scheme. A Level 1 organization is considered operating in chaos, with a Level 5 organization considered a highly tuned operation. Among US Government and particularly US Defense Department contractors the model has been widely implemented over the past decade due to government procurement mandates. It is less commonly followed in the commercial software sector, though it has made some inroads there as well. See chapter endnote #67 for details.*
**Platform Drivers**

- **Execution Time Constraint (TIME):** Measure of the execution time constraint imposed upon a software system.

- **Main Storage Constraint (STOR):** Measures the degree of main storage constraint imposed on a software system or subsystem.

- **Platform Volatility (PVOL):** Measure of the degree of volatility/rate of change in the complex of hardware and software (operating system, DBMS, etc.) that the product under development calls upon to perform its tasks.

**Personnel Drivers**

- **Analyst Capability (ACAP):** Analysts are personnel that work on requirements, high level design, and detailed design.

- **Programmer Capability (PCAP):** Measure of the capability of the programmers as a team rather than as individuals, and considers ability, efficiency, thoroughness, and the ability to communicate and cooperate.

- **Personnel Continuity (PCON):** Measure of the development project’s annual personnel turnover rate.

- **Applications Experience (APEX):** Measure of the project team’s overall level of experience building the current type of product under development.

- **Platform Experience (PLEX):** Measures the project team’s experience with modern and powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities.
- **Language and Tool Experience (LTEX):** Measure of the level of programming language and software tool experience of the project team.

**Project Drivers**

- **Use of Software Tools (TOOL):** Measure of the extent advanced software development tools are used during development.

- **Multi-site Development (SITE):** Measure of the nature of project development site locations (from fully collocated to international distribution), and communication support between those sites (from surface mail and phone access to full interactive multimedia).

- **Required Development Schedule (SCED):** Measure of the schedule constraint imposed on the project; defined in terms of the percentage schedule stretch-out or acceleration with respect to a nominal schedule for a project requiring a given amount of effort.

In practical terms, the application of each of the scale factors and effort multipliers described above requires an estimator using COCOMO II to rate each parameter on a scale from extra low to extra high, based upon unique sets of criteria that have been determined for each item.
Table 2.1a - COCOMO II.2000 Scale Driver Rating Criteria

<table>
<thead>
<tr>
<th>Scale Drivers</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREC</td>
<td>thoroughly unprecedented</td>
<td>largely unprecedented</td>
<td>Somewhat unprecedented</td>
<td>generally familiar</td>
<td>largely familiar</td>
<td>thoroughly familiar</td>
</tr>
<tr>
<td>FLEX‡</td>
<td>rigorous</td>
<td>occasional relaxation</td>
<td>some relaxation</td>
<td>general conformity</td>
<td>some conformity</td>
<td>general goals</td>
</tr>
<tr>
<td>RESL‡</td>
<td>little (20%)</td>
<td>some (40%)</td>
<td>Often (60%)</td>
<td>generally (75%)</td>
<td>mostly (90%)</td>
<td>full (100%)</td>
</tr>
<tr>
<td>TEAM</td>
<td>very difficult interactions</td>
<td>some difficult interactions</td>
<td>basically cooperative interactions</td>
<td>largely cooperative</td>
<td>highly cooperative</td>
<td>seamless interactions</td>
</tr>
<tr>
<td>PMAT</td>
<td>SW-CMM Level 1 Lower</td>
<td>SW-CMM Level 1 Upper</td>
<td>SW-CMM Level 2</td>
<td>SW-CMM Level 3</td>
<td>SW-CMM Level 4</td>
<td>SW-CMM Level 5</td>
</tr>
</tbody>
</table>

*° degree of required adherence to stated system requirements.

‡ percentage of module interfaces specified subjectively averaged with the percentage of known significant risks mitigated.

or the estimated Process Maturity Level
### Table 2.1b - COCOMO II.2000 Post-Architecture Effort Multiplier Rating Criteria

<table>
<thead>
<tr>
<th>Cost Drivers</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELY</td>
<td>slight inconvenience</td>
<td>low, easily recoverable losses</td>
<td>Moderate, easily recoverable losses</td>
<td>high financial loss</td>
<td>risk to human life</td>
<td></td>
</tr>
<tr>
<td>DATA</td>
<td>(DB bytes / Pgm SLOC) &lt; 10</td>
<td>10 ≤ D/P &lt; 100</td>
<td>100 ≤ D/P &lt; 1000</td>
<td>D/P &gt; 1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPLX</td>
<td>See Table 2.1c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RUSE</td>
<td>none</td>
<td>across project</td>
<td>across program</td>
<td>across product line</td>
<td>across multiple product lines</td>
<td></td>
</tr>
<tr>
<td>DOCU</td>
<td>many life cycle needs uncovered</td>
<td>some life cycle needs uncovered</td>
<td>correct amount for life cycle needs</td>
<td>excessive for life cycle needs</td>
<td>very excessive for life cycle needs</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>≤ 50% use of available execution time</td>
<td>70% use</td>
<td>85% use</td>
<td>95% use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STOR</td>
<td>≤ 50% use of available storage</td>
<td>70% use</td>
<td>85% use</td>
<td>95% use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PVOL</td>
<td>major change every 12 mo.; minor change every 1 mo.</td>
<td>major: 6 mo.; minor: 2 wk.</td>
<td>major: 2 mo.; minor: 1 wk.</td>
<td>major: 2 wk.; minor: 2 days</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACAP</td>
<td>15th percentile</td>
<td>35th percentile</td>
<td>55th percentile</td>
<td>75th percentile</td>
<td>90th percentile</td>
<td></td>
</tr>
<tr>
<td>PCAP</td>
<td>15th percentile</td>
<td>35th percentile</td>
<td>55th percentile</td>
<td>75th percentile</td>
<td>90th percentile</td>
<td></td>
</tr>
<tr>
<td>PCON</td>
<td>48% / year</td>
<td>24% / year</td>
<td>12% / year</td>
<td>6% / year</td>
<td>3% / year</td>
<td></td>
</tr>
<tr>
<td>APEX</td>
<td>≤ 2 months</td>
<td>6 months</td>
<td>1 year</td>
<td>3 years</td>
<td>6 years</td>
<td></td>
</tr>
<tr>
<td>PLEX</td>
<td>≤ 2 months</td>
<td>6 months</td>
<td>1 year</td>
<td>3 years</td>
<td>6 years</td>
<td></td>
</tr>
<tr>
<td>LTEX</td>
<td>≤ 2 months</td>
<td>6 months</td>
<td>1 year</td>
<td>3 years</td>
<td>6 years</td>
<td></td>
</tr>
<tr>
<td>TOOL</td>
<td>edit, code, debug</td>
<td>simple, frontend, backend CASE, little integration</td>
<td>Basic life cycle tools, moderately integrated</td>
<td>strong, mature life cycle tools, moderately integrated</td>
<td>strong, mature, proactive life cycle tools, well integrated with processes, methods, reuse</td>
<td></td>
</tr>
<tr>
<td>SITE: Collocation</td>
<td>international</td>
<td>multi-city and multi-company</td>
<td>multi-city or multi-company</td>
<td>same city or metro area</td>
<td>same building or complex</td>
<td>fully collocated</td>
</tr>
<tr>
<td>SITE: Communication</td>
<td>some phone, mail</td>
<td>individual phone, FAX</td>
<td>narrow-band electronic communication</td>
<td>wide-band elect. comm., occasional video conf.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCED</td>
<td>75% of nominal</td>
<td>85% of nominal</td>
<td>100% of nominal</td>
<td>130% of nominal</td>
<td>160% of nominal</td>
<td></td>
</tr>
</tbody>
</table>
### Table 2.1c - CPLX Rating Criteria

<table>
<thead>
<tr>
<th>Ratings</th>
<th>Control Operations</th>
<th>Computational Operations</th>
<th>Device-dependent Operations</th>
<th>Data Management Operations</th>
<th>User Interface Management Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Very Low</strong></td>
<td>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.</td>
<td>Evaluation of simple expressions: e.g., $A=B+C*(D-E)$</td>
<td>Simple read, write statements with simple formats.</td>
<td>Simple arrays in main memory. Simple COTS-DB queries, updates.</td>
<td>Use of simple graphic user interface (GUI) builders.</td>
</tr>
<tr>
<td><strong>Low</strong></td>
<td>Straightforward nesting of structured programming operators. Mostly simple predicates.</td>
<td>Evaluation of moderate-level expressions: e.g., $D=\text{SQRT}(B^{**2}-4<em>A</em>C)$</td>
<td>No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level.</td>
<td>Single file subsetting with no data structure changes, no edits, no intermediate files. Moderately complex COTS-DB queries, updates.</td>
<td>Simple use of widget set.</td>
</tr>
<tr>
<td><strong>Nominal</strong></td>
<td>Mostly simple nesting. Some intermodule control. Decision tables. Simple callbacks or message passing, including middleware-supported distributed processing.</td>
<td>Use of standard math and statistical routines. Basic matrix/vector operations.</td>
<td>I/O processing includes device selection, status checking and error processing.</td>
<td>Multi-file input and single file output. Simple structural changes, simple edits. Complex COTS-DB queries, updates.</td>
<td>HRT support and use of advanced event-driven system.</td>
</tr>
</tbody>
</table>

Note - the final overall rating for CPLX is determined as a subjective average of the ratings given to each of the individual factors contributing to the software complexity identified by the headings of the five right-most columns.
<table>
<thead>
<tr>
<th>Ratings</th>
<th>Control Operations</th>
<th>Computational Operations</th>
<th>Device-dependent Operations</th>
<th>Data Management Operations</th>
<th>User Interface Management Operations</th>
</tr>
</thead>
</table>
Table 2.2 - COCOMO II.2000 Scale Factor and Effort Multiplier Numeric Values

| Driver | Symbol | XL  | VL  | L   | N   | H   | VH  | XH  | Productivity Range
|--------|--------|-----|-----|-----|-----|-----|-----|-----|------------------
|        |        |     |     |     |     |     |     |     |                  |
| PREC   | SF₁    | 6.20| 4.96| 3.72| 2.48| 1.24| 0.00|     | 1.33             |
| FLEX   | SF₂    | 5.07| 4.05| 3.04| 2.03| 1.01| 0.00|     | 1.26             |
| RESL   | SF₃    | 7.07| 5.65| 4.24| 2.83| 1.41| 0.00|     | 1.39             |
| TEAM   | SF₄    | 5.48| 4.38| 3.29| 2.19| 1.10| 0.00|     | 1.29             |
| PMAT   | SF₅    | 7.80| 6.24| 4.68| 3.12| 1.56| 0.00|     | 1.43             |

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Post-Architecture Effort Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELY</td>
<td>EM₁</td>
<td>0.82</td>
<td>0.92</td>
<td>1.00</td>
<td>1.10</td>
<td>1.26</td>
<td></td>
<td></td>
<td>1.54</td>
</tr>
<tr>
<td>DATA</td>
<td>EM₂</td>
<td>0.90</td>
<td>1.00</td>
<td>1.14</td>
<td>1.28</td>
<td></td>
<td></td>
<td></td>
<td>1.42</td>
</tr>
<tr>
<td>CPLX</td>
<td>EM₃</td>
<td>0.73</td>
<td>0.87</td>
<td>1.00</td>
<td>1.17</td>
<td>1.34</td>
<td>1.74</td>
<td></td>
<td>2.38</td>
</tr>
<tr>
<td>RUSE</td>
<td>EM₄</td>
<td>0.95</td>
<td>1.00</td>
<td>1.07</td>
<td>1.15</td>
<td>1.24</td>
<td></td>
<td></td>
<td>1.31</td>
</tr>
<tr>
<td>DOCU</td>
<td>EM₅</td>
<td>0.81</td>
<td>0.91</td>
<td>1.00</td>
<td>1.11</td>
<td>1.23</td>
<td></td>
<td></td>
<td>1.52</td>
</tr>
<tr>
<td>TIME</td>
<td>EM₆</td>
<td>1.00</td>
<td>1.11</td>
<td>1.29</td>
<td>1.63</td>
<td></td>
<td></td>
<td></td>
<td>1.63</td>
</tr>
<tr>
<td>STOR</td>
<td>EM₇</td>
<td></td>
<td></td>
<td>1.00</td>
<td>1.05</td>
<td>1.17</td>
<td>1.46</td>
<td></td>
<td>1.46</td>
</tr>
<tr>
<td>PVOL</td>
<td>EM₈</td>
<td>0.87</td>
<td>1.00</td>
<td>1.15</td>
<td>1.30</td>
<td></td>
<td></td>
<td></td>
<td>1.49</td>
</tr>
<tr>
<td>ACAP</td>
<td>EM₉</td>
<td>1.42</td>
<td>1.19</td>
<td>1.00</td>
<td>0.85</td>
<td>0.71</td>
<td></td>
<td></td>
<td>2.00</td>
</tr>
<tr>
<td>PCAP</td>
<td>EM₁₀</td>
<td>1.34</td>
<td>1.15</td>
<td>1.00</td>
<td>0.88</td>
<td>0.76</td>
<td></td>
<td></td>
<td>1.76</td>
</tr>
<tr>
<td>PCON</td>
<td>EM₁₁</td>
<td>1.29</td>
<td>1.12</td>
<td>1.00</td>
<td>0.90</td>
<td>0.81</td>
<td></td>
<td></td>
<td>1.59</td>
</tr>
<tr>
<td>APEX</td>
<td>EM₁₂</td>
<td>1.22</td>
<td>1.10</td>
<td>1.00</td>
<td>0.88</td>
<td>0.81</td>
<td></td>
<td></td>
<td>1.51</td>
</tr>
<tr>
<td>PLEX</td>
<td>EM₁₃</td>
<td>1.19</td>
<td>1.09</td>
<td>1.00</td>
<td>0.91</td>
<td>0.85</td>
<td></td>
<td></td>
<td>1.40</td>
</tr>
<tr>
<td>LTEX</td>
<td>EM₁₄</td>
<td>1.20</td>
<td>1.09</td>
<td>1.00</td>
<td>0.91</td>
<td>0.84</td>
<td></td>
<td></td>
<td>1.43</td>
</tr>
<tr>
<td>TOOL</td>
<td>EM₁₅</td>
<td>1.17</td>
<td>1.09</td>
<td>1.00</td>
<td>0.90</td>
<td>0.78</td>
<td></td>
<td></td>
<td>1.50</td>
</tr>
<tr>
<td>SITE</td>
<td>EM₁₆</td>
<td>1.22</td>
<td>1.09</td>
<td>1.00</td>
<td>0.93</td>
<td>0.86</td>
<td>0.80</td>
<td></td>
<td>1.53</td>
</tr>
<tr>
<td>SCED</td>
<td>EM₁₇</td>
<td>1.43</td>
<td>1.14</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td></td>
<td></td>
<td>1.43</td>
</tr>
</tbody>
</table>

For Effort Calculations:
- Multiplicative constant A = 2.94;
- Exponential constant B = 0.91

For Scale Factors:
\[
PR_{SF}^{n} = \frac{(100)^{0.91 + (0.01 \times SF_{n_{max}})}}{(100)^{0.91}}
\]

For Effort Multipliers:
\[
PR_{EM}^{n} = \frac{EM_{n_{max}}}{EM_{n_{min}}}
\]
These ratings correspond to specific numerical factors that are then applied in the model to derive the final estimated software development effort. It is these all

---

"Prediction Accuracy Level" refers to the percentage of estimates produced by the model during validation that came within (+/-) 20% of actual reported values—PRED (.20); within (+/-) 25% of actuals—PRED (.25); and within (+/-) 30% of actuals—PRED (.30); note that this is NOT the same concept as an X% confidence interval built around a given point estimate based upon a standard deviation, but this is the industry accepted convention for discussing the "accuracy" of software estimation models. Also, it should be noted that the results reported in table 2.3 are based upon control or validation data that was also included as part of the calibration data (another common though statistically undesirable practice when building these kinds of models, but often necessitated by the difficulty in collecting large quantities of acceptable historical software project data—see end reference #98); when validated against a randomly chosen independent data set NOT included as part of the calibration data, COCOMO II.2000 produced a PRED (.30) result of 69% as opposed to the 75% reported above before stratification, but which is still considered a good result for these kinds of models (see chapter 1 endnote #11, p. 173).

"Before Stratification" refers to how well the model did when testing control data against the generic calibration that included all 161 available historical project data points. "After Stratification" refers to how well the model did when testing subsets of the control data against calibrations that were performed only against data that came from the same organization as a given control data set. In other words, table 2.3 supports the long held notion (see chapter endnotes #72 and #73) that models calibrated against an organization's own internal data will usually perform better for that organization than models calibrated to a generic set of data coming from many different organizations (though there is at least one recent study that strongly contests this belief—see chapter endnote #75); and again, keep in mind that the control data spoken of here used for validation was included among the data used for calibration both before and after stratification (see the preceding footnote above).
important numeric values (see table 2.2) that are the end result of the lengthy Bayesian-based model calibration process that was mentioned in section 1.1.

Tables 2.1a through 2.1c on the preceding pages list the criteria for rating each of the COCOMO II scale factors and effort multipliers, while again, table 2.2 shows the corresponding numeric parameter values for each of those ratings for the 2000 calibration of COCOMO II based on 161 historical project data points. Finally, table 2.3 indicates the percentage of projects in the data that had their effort and schedule estimates (schedule modeling is discussed in the following section) come within the indicated percentage (i.e., PRED level) of their actual reported values after calibration.

2.4.2 Schedule Estimation

Effort estimation is only half of what COCOMO does. Of equal importance is the schedule estimation capability it provides. As with the effort model, the COCOMO II schedule model builds on a standard form:

\[
\text{Schedule} = C \cdot (\text{Effort})^D
\]

Eq. 2.10

where

\[
\text{Schedule} = \text{software development calendar duration (usually given in months)}.
\]

\[
\text{Effort} = \text{estimated software development effort (usually given in person-months)}.
\]

\[
C = \text{a multiplicative conversion constant relating software development effort to development schedule}.
\]
D = an exponential constant usually < 1 that reflects the fact that development schedule (unlike effort) does generally exhibit nonlinear economies of scale (e.g., doubling the development effort will less than double the schedule).

The COCOMO II Post-Architecture schedule model makes the following adjustments:

\[
\text{Schedule}_{\text{COC}} = C_{\text{COC}} \left( \text{Effort}_{\text{COC-NS}} \right)^{F_{\text{COC}}} \cdot (\text{SCED}\% / 100) \quad \text{Eq. 2.11a}
\]

where

\[
F_{\text{COC}} = D_{\text{COC}} + 0.2 \cdot (0.01 \sum_{j=1}^{5} S_{F_{\text{COC}(j)}}) \quad \text{Eq. 2.11b}
\]

and

\[
\text{Schedule}_{\text{COC}} = \text{software development calendar duration (in months)}.
\]

\[
\text{Effort}_{\text{COC-NS}} = \text{COCOMO II estimated software development effort (in person-months)} \text{ with the schedule effort multiplier (EM}_{\text{COC}(17)}: \text{SCED}) \text{ set to nominal (meaning a numerical value of 1.0)}
\]

\[
\text{SCED}\% = \text{required percentage of schedule compression relative to a nominal schedule (maximum compression allowed is down to 75\% of a nominal schedule)}.
\]

\[
C_{\text{COC}} = \text{a multiplicative conversion constant relating software development effort to development schedule}.
\]

\[
F_{\text{COC}} = \text{an exponential factor that accounts for the fact that development schedule does generally exhibit nonlinear economies of scale and which in turn is now a function of a constant } D_{\text{COC}} \text{ and the five scale factors } S_{F_{\text{COC}(j)}}.
\]

* As a broad rule of thumb, software development projects using standard—as opposed to, for example, rapid application development or "RAD" development techniques—tend to follow a cube root rule when relating effort to schedule (see chapter 1 endnote #1, pp. 88-89).
\[ SF_{\text{coc}(q)} = \text{the same five "scale factors" used in the Post-Architecture effort model characterizing project conditions and that have been shown to also have nonlinear impacts on software development calendar duration just as with development effort; in this case, however, they determine not whether economies of scale will likely present during the development with regard to calendar schedule, but rather how large those economies of scale will likely be.} \]

The 2000 calibration of COCOMO II currently assigns a value of 3.67 to \( C_{\text{coc}} \) and 0.28 to \( D_{\text{coc}} \) (see table 2.2). Look again at table 2.3 showing the predictive accuracy achieved by the 2000 calibration. Note that the accuracy achieved by COCOMO II.2000 for schedule prediction is slightly less than that achieved for effort prediction, which means there are probably either hidden issues yet unresolved with respect to the conditioning of the data used to calibrate the model (see section 3.2.3), or perhaps equally likely there are other factors impacting schedule not yet being captured by the schedule model as currently formulated.

To put these numbers in perspective, however, and to complete this review of COCOMO II, it should be noted that software estimation models that consistently come within 30% of actual reported values 70% of the time are generally considered by the professional estimation community to be good models performing about as well as can be expected within this field.\(^7\)

\(^7\) This may come as a surprise to some readers who are used to dealing with engineering precisions of much tighter tolerance, particularly as typically found in the hardware or electronic worlds. Designing software, however, is different. Even with the drive to construct new software systems out of commercially available pre-built components, major software systems to a large degree are still "one off" kinds of things, with very high levels of internal complexity. This makes it very difficult to establish quantifiable standards for the amount of effort it will likely take to produce a given amount
2.5 Chapter 2 Endnotes

1 Internet home page for the SEI CBS Initiative is at http://www.sei.cmu.edu/cbs/.


4 Internet home page for the SPC is at http://www.software.org.


12 Internet home page for RAND Corporation is at http://www.rand.org.

of software functionality. Thus the software community lives with estimation accuracies as noted above because there are no better alternatives.


34 Boehm et al., "Software Development Cost Estimation Approaches - A Survey."


39 Madachy, R., instructor's class notes, course CSCI 577a, Dept. of Computer Science, University of Southern California, Los Angeles, CA, Fall term, 1999.


41 Forrester, J., Industrial Dynamics.


68 Adapted from Boehm et al., *Software Cost Estimation with COCOMO II*, inside front cover.

69 Adapted from Boehm et al., *Software Cost Estimation with COCOMO II*, inside front cover.

70 Adapted from Boehm et al., *Software Cost Estimation with COCOMO II*, p. 43.

71 Adapted from Boehm et al., *Software Cost Estimation with COCOMO II*, inside front cover.


3. Research Approach

3.1 Setting the Context

Section 3 describes the overall approach used to develop the COCOTS model. The work began with a preliminary study completed in June of 1997 (again, see chapter 1 endnote #18), demonstrating a sort of "proof of concept." Since that time, the COTS integration cost model defined in that first 1997 study has been further refined—including acquiring the name COCOTS—and historical project data from a number of industrial sources suitable for calibration purposes has been collected.

As will be elaborated upon in the following sections, figure 3.1 illustrates the formal steps in the complete process that ideally would be applied to arrive at a calibrated CBS estimation model. However, only the boxes of steps one through five indicate those activities that were actually able to be accomplished for this thesis. Due to the limited number of calibration data points available as of this writing relative to the number of parameters in the model, the boxes of steps six and seven indicate those modeling activities that had to left for future investigation.

The issue is statistical degrees of freedom. A rule-of-thumb accepted within the software modeling community is that to produce a reasonable statistical calibration, you should have at least five times the number of calibration data points for every
parameter appearing in a model. For example, as discussed in section 3.2.4, Devnani-Chulani used 161 data points to perform her Bayesian calibration of the 24 parameters in COCOMO II, thus exceeding by 34% the minimum 120 points suggested by the rule-of-thumb. Since COCOTS as currently formulated has some 20 parameters, the rule suggests there should be at least 100 data points available. As of now, though, there are only 20 historical project data points available for use in calibration. The reader can see the problem. As a result, attempts at a Bayesian-based
calibration of COCOTS must be postponed until such time as there are sufficient
data points available. In the meantime, the final calibration presented in this report
follows the same form as those done since the initial 1997 study; that is, the
calibration that is normally the output of step six was based upon simple data
regression and expert judgment as opposed to Bayesian statistical means—simple
regression on the single multiplicative conversion factor A in the glue code
submodel (see equation 4.5a), expert judgment for the other parameters in that same
equation; for the parameters in the assessment and tailoring submodels, simple
averaging.

An important feature of the modeling methodology illustrated in figure 3.1 is the
feedback loops emanating from step seven. Even though to date calibrations
performed during passes through step seven have been based on simple regression as
opposed to Bayesian methods, the value of those feedback loops still holds as
follows: insights gained from that first 1997 model were used as feedback in a return
to step three which resulted in a second, more refined COTS model (again, see
chapter 1 endnote #20). The insights gained from that second model in turn led to yet
another loop back through step three and an even more radical revision of the model
(see chapter 1 endnote #21). This third version of the model was iterated upon once
again with a fourth pass through steps five through seven, leading to the fourth and
final iteration of the model presented in this thesis.

The first iteration model was essentially an attempt at proof-of-concept. It
developed initial definitions for model parameters but was calibrated to a set of
carefully controlled student projects. The second iteration model moved from the student to the real world. Its parameter definitions were refined based on further interaction with industry practitioners and its calibration was based on actual industry projects. The results of that calibration made it clear the model was not adequately accounting for particular kinds of COTS-related effort. This led to a radical revision of the model in the third iteration. The third iteration model split the existing model into four separate submodels in an attempt to capture those other sources of effort. The fourth and final iteration represents a simplification of the previous model, reducing the number of submodels from four to three and further clarifying parameter definitions. (At some point in the future, a fifth iteration model based upon a Bayesian calibration will ideally be produced.)

3.2 USC-CSE Seven Step 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 (see section 2.2.4), 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, a multi-step modeling methodology has been adopted (see chapter 1 endnote #12) that is useful for developing software estimation models when the amount of related empirical data is initially minimal. Using Bayesian statistical techniques, this approach allows one 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. Again, illustrated in figure 3.1, this method allows one to go forward with developing reasonable model definitions until such time as significant empirical data becomes available for further model refinement and calibration.

Beginning with step one, a standard literature search is performed to determine relevant background information and to assess the status and achievements of any existing related estimation models. This sets the stage for step two, behavioral analysis, during which insights gained from the information learned in step one are coupled with consultations and workshops conducted with domain experts in an attempt to identify the most significant and effort-intensive activities (not necessarily the same thing) that individuals typically engage in while developing large software systems. Other items potentially affecting cost or effort are also examined at this stage, but the focus is always on those activities and factors that are most germane to the particular software development methods or techniques that are being modeled.
Next comes step three, wherein the factors impacting effort examined in step two are recast into formally defined potential cost drivers. Mathematical relationships between those drivers are also postulated. It is at this point that a candidate cost model really first appears. Step four brings together another panel of domain experts in a Delphi exercise (see sections 2.2.1 and 3.2.2) to establish initial numerical values for the parameters defined in step three. Historical empirical project data addressing the cost drivers defined in step three is then gathered (step five) and used to generate a revised set of numerical values for the parameters in the model using Bayesian statistical techniques (step six). Finally, in step seven, the results of the model calibration produced in step six are examined, leading to revisions of the model and the collection of more empirical data in attempts to improve its fidelity.

This iteration on the model in terms of its structure and to grow the historical project database used for its calibration can be repeated as often as necessary or as long as it makes economic sense to do so.

Now that the general modeling process has been described, the next few sections will summarize how the various steps in figure 3.1 have been worked through to arrive at the current fourth iteration COCOTS model.

3.2.1 Steps 1 through 3: Preliminary Model Definition

When beginning this research, the first challenge was to recognize that COTS products serve different roles and are not all of a kind. COTS products can be used in
essentially three ways: 1) as a component of a tool bed, 2) as a component of a system development or support infrastructure, and 3) as a component of a new application.

Currently the feeling is that the issue of COTS software being integrated as infrastructure or as part of a tool bed can be addressed fairly adequately within the COCOMO II model itself via the following drivers (see section 2.4.1):

- **COTS as infrastructure**: Platform Volatility (PVOL) and developer Platform Experience (PLEX).

- **COTS as tools**: Use of Software Tools (TOOL) and developer Language & Tool Experience (LTEX).

Modeling the impact of COTS components used as tools or infrastructure using the above-named COCOMO II drivers is not perfect, because these drivers do not address such issues as licensing or assessment or some of the other factors contributing to cost that were indicated previously (see section 1.2). Nonetheless, they can still capture much of the effort associated with using COTS components in these contexts.

The problem which remains totally outside the bounds of COCOMO, however, is that depicted in the upper half of the left side of figure 3.2, in which black box COTS components are being integrated as part of an application (as for white box COTS
components, see again the second footnote in section 1.2). It is primarily this problem which is being addressed by COCOTS, at least for the purposes of this thesis.

*COTS Modeling Problem Context*

*(COTS Components as Application Elements, Infrastructure, or Tools)*

![Diagram](image)

Cost Modeling Currently Addressed Within COCOMO II: COTS as Infrastructure and Tools

Figure 3.2 - COTS component role being addressed by COCOTS: COTS as application elements.

Having bounded the problem as described above, the first formal step taken in developing COCOTS as shown in figure 3.1 was to conduct a general literature review, the results of which were discussed in section 2.1. To recap, however, much qualitative information was found addressing the use of COTS products in software development, but very little information was available related to quantitative estimation models.
One source of relevant information that proved to be very helpful at this stage, however, was a database maintained at the Software Engineering Institute in Pittsburgh. Called the Software Engineering Risk Repository (SERR), it consists of a series of statements solicited by software professionals from around the country regarding software engineering issues. These statements are grouped by topic and then parsed by content. The statements can then be examined by an automated tool developed by the SEI to summarize the available information in the form of lexical maps. Applying this tool on a subset of the statements in the Risk Repository that concerned the use of COTS components, I was able to create a series of graphs visually emphasizing the relative importance of the key concerns relating to COTS integration which were to be found in the professionals' statements. The information extracted from the SERR in this manner became the first part of the kernel of ideas that ultimately lead to the concepts captured in the cost drivers currently found in COCOTS.

(For more detail on the SERR database and the lexical maps discussed above, see section V1.A and appendix C of the 1997 COTS software study, chapter 1 endnote #18.)

A synthesis of the ideas gleaned from the literature review, the Ellis model (section 2.3.3), the COCOMO model (section 2.4), and the SERR database lead to the creation of a preliminary data collection survey. The purpose of this was to conduct a first round of data collection within industry with the goal of prioritizing a candidate set of COTS product integration effort influence factors. About 800
surveys were sent out with around a 5% return rate. The results (detailed in section VI.D and appendix D2 of chapter 1 endnote #18) of this survey, along with the other information on hand about COCOMO II and the Ellis Model, where then used as a basis by a panel of industry experts for helping to refine those candidate effort influence factors into the cost drivers which appeared in the first pass COTS estimation model (see section VI.E of chapter 1 endnote #18). This represented the completion of step two in the modeling process shown in figure 3.1.

Moving on to step three, insights garnered from the Ellis model and that first industry panel suggested that the prime source of COTS-associated effort was the activity needed to create the software needed to bind or integrate a COTS product into the larger software system within which it is being used. Because of this, for the first pass COTS estimation model it seemed to make sense to adopt a similar mathematical form as that of COCOMO II, which focuses on the creation of new application software. The COTS model, however, would focus on the creation of COTS glue code.

The definition of the first pass model was now complete. The next step was the derivation of initial numerical values for the model parameters.

3.2.2 Step 4: Delphi Experiment

The Delphi technique and its history were discussed earlier in section 2.2.1. To review, however, it is a means of guiding a group of informed individuals to a
consensus of opinion on some issue. Participants are asked to make some assessment regarding an issue individually in a preliminary round, without consulting the other participants in the exercise. The first round results are then collected, tabulated, and then returned to each participant for a second round, during which the participants are again asked to make an assessment regarding the same issue, but this time with knowledge of what the other participants did in the first round. The second round usually results in a narrowing of the range in assessments by the group, pointing to some reasonable middle ground regarding the issue of concern.

This is a useful technique for coming to some conclusion regarding an issue when the only information available is based more on "expert opinion" than hard empirical data, and thus appears as the fourth step in figure 3.1. However, for the purposes of software estimation model building, a variant on the standard Delphi technique called Wideband Delphi is preferred. Given the highly technical nature of the model building task, standard Delphi does not allow enough information to flow between the participants for them to all necessarily get a sufficient grasp of the issues involved. Thus, under Wideband Delphi, rather than remaining totally anonymous from each other, the participants in a Delphi experiment are brought together to discuss the issue at hand during both rounds, but their individual responses to the question of concern are still kept anonymous. This has the advantage of preserving the benefits of both face-to-face interchange and anonymous decision making.

As applied to COCOTS, Wideband Delphi was used as means of obtaining consensus on what might be reasonable initial values for the thirteen effort multiplier
parameters that appeared in the first pass COTS estimation model. These Delphi derived parameter values were then later used as a starting point for the model calibration activity that is the subject of step six in figure 3.1.*

Again, the details of how the Delphi technique was used to derive initial parameter values for the first pass model are given in sections VII.E and VII.F of chapter 1 endnote #18 while appendix D1 of that same document contains the Delphi experiment survey instrument.

3.2.3 Step 5: Data Collection & Conditioning

The next step in the development of the first pass COTS model was the collection of empirical data. Specifically, actual effort and sizing data was needed for past software projects that incorporated COTS products in their design. This information was necessary to help calibrate the model and validate its form.

To that end, a new data collection survey was drafted asking for specific project level information, including effort and sizing data associated with the glue code needed to integrate any COTS components. In addition, just as shown for COCOMO II in tables 2.1a through 2.1c in section 2.4.1, the survey asked each cost driver defined for the first pass model to be rated on a five point scale from very low

* After the redefinition of parameters that occurred with the second pass model, a new Delphi experiment was not conducted. Instead, initial pre-calibration parameter values for the revised model were derived based on extrapolations from the first pass model Delphi results (see chapter 1 endnote #20). Moreover, the second pass model in turn became just one of four submodels in the third pass model, namely the COCOTS glue code submodel. Given the nature of the other submodels in the third pass model (detailed in section 5.3), Delphi experiments do not seem to be of much use for determining initial parameter values in these cases and so as of this writing have not been performed for any model other than the glue code submodel.
through very high according to the development conditions that obtained during the life of the project.

This follow-on survey was then used to collect detailed project information from both completed industrial and carefully controlled graduate student software projects. Ultimately, it was the student data that was used as the basis for the calibration of the first pass model. The industrial data that was available that first time out proved to be too problematic to be usable for calibration purposes—which brings me to the topic of "data conditioning."

As in any research endeavor, the data collected for the calibration of COCOTS cannot be used simply "as is." Each data point must go through a significant vetting and conditioning process, to make sure it is appropriate for inclusion in the calibration. Data conditioning is an essential activity in any data collection and analysis process. Even when people try to provide the best data they can, there are a number of known problems and subtle sources of misunderstanding that can inject bias into their data. Use of such data to calibrate cost models can lead to erroneous results should these and other sources of data contamination not be removed.

Some of the problems with the industrial data gathered for the first pass model were of the nature as those problems cautioned about beginning on page 22 in section 2.2.4 regarding the requirements for data that are to be used for statistical regression. Other issues were among those commonly associated with software data collection and sources of data contamination:
1. **Inconsistent definitions** - COCOTS sometimes defines terms slightly differently than similar terms used in its parent model COCOMO II. For example, the REVL term in COCOMO II and the CREVL term in COCOTS both address the impact of requirements evolution on the amount of newly developed code that may have to be reworked before final delivery. However, the CREVL term also includes the impact of volatility in any COTS components being used whereas the REVL term does not. Moreover, COCOMO II defines terms differently than previous versions of COCOMO. For example, COCOMO II and COCOTS use SLOC (Source Lines of Code) instead of DSI (Delivered Source Instructions) which were used in the original COCOMO model. An "IF-THEN-ELSE, ELSE IF" pair will now count as a single SLOC instead of two DSI when a terminal semi-colon is used for the counting conventions. As another example, COCOMO II and COCOTS use 152 person hours per person-month and assume casual overtime is not included as part of the burden. If something different were used, the models would generate erroneous answers.

2. **Improper scope** - COCOTS and COCOMO II assume that the project's scope includes certain activities and excludes others. For example, software testing is included while software support to system integration and test is not. As another example, software documentation that is normally generated during the software development life cycle is included while customer unique documentation is not.
Again, one would generate erroneous answers if the model is used outside of its proper scope.

3. **Double Counting** - Sometimes items are double counted or taken into account twice using several factors within the model. For example (as indicated above), REVL and CREVL are used to take into account volatile requirements. However, some people double dip by improperly rating the Precededness or Architecture/Risk Resolution scale factors lower than they should be to take volatility into account. One needs to understand what the factor ratings involve prior to rating them to avoid making this mistake.

4. **Averaging** - Often, people use average ratings for groupings that extend across subsystems and the project. Because they haven't taken the time to get into the details, they consolidate their estimate and lose fidelity because little differentiation is made between different types of software. One can avoid this problem and greatly improve the accuracy of the estimates by breaking down the project into finer grained components.

5. **Garbage In, Garbage Out** - Another common problem is the use of erroneous assumptions. People often use models to generate quick-and-dirty estimates. They make all sorts of simplifying assumptions in their quest for numbers. One way to avoid problems of this sort is to take a little more time to develop
realistic, but simplifying assumptions. This often takes some interaction with both the developer and customer communities.

6. **Observational Bias** - Finally, many people tend to be overly optimistic/pessimistic when they estimate. Biases either way should be avoided especially when they can become a self-fulfilling prophecy. Use of Wideband Delphi in which groups of experts reach consensus on their estimates reduces such biases. However, such group estimates take more time to achieve and may not be practical under some circumstances.

To avoid introducing data with such problems into a model calibration, the data vetting and conditioning process is used. This conditioning process includes several activities.

First, the original data collection instruments (in my case, completed COCOTS software project data reporting surveys) need to be examined for completeness and consistency. This means screening each data survey to identify missing, unreasonable and inconsistent entries. For example, a large reported effort (in person-months) for a small sized application needs to be checked. As another example, reporting significant prior similar applications experience in the form of one COCOTS cost driver for an application reported as being highly unprecedented via another COCOTS cost driver seems inconsistent. Verifying such data entries
would require going back and contacting the individual who originally supplied the data.

Next, the general "reasonableness" of the data can be determined by comparing it to published benchmarks. This provides yet another check on the data's general validity. Any great variation of a data point from published benchmarks will require further investigation to see whether that variation can be explained. If it can't be explained, that raises questions about the suitability of that data point for inclusion in the calibration.

This process ultimately forced the first pass model industrial data to be judged unsuitable for calibration purposes. The main issue was that the data was simply too old and incomplete. However, insights gained from the examination of this initial industrial data led to significant improvements in how I went about gathering industrial data for the subsequent versions of COCOTS. As a result of these improvements, there are now some 20 data points (information on historical software projects using COTS components) in the COCOTS calibration database. This data is the result of 200+ hours of on-site data collection interviews conducted with knowledgeable project personnel, usually a project manager or technical lead.

The original calibration data collection survey for the first pass model can be found in appendix D3 of chapter 1 endnote #18. The most recent version of that survey can be found in appendix B at the end of this thesis.
3.2.4 Step 6: Bayesian-assisted Model Calibration

(Recall that due to data limitations a Bayesian calibration has not been performed. For the sake of completeness, however, in describing the whole seven step process that would ideally be followed, the technique is summarized here.)

The next step in the model building process shown in figure 3.1 is to derive a calibration of the proposed model that combines the judgment of domain experts with objective historical project data using Bayesian statistical means. It was shown by Devnani-Chulani during the development of COCOMO II that this approach can lead to a model with a better generalized ability to estimate software costs than one based on empirical data or expert judgment alone.

The reason is due to a limitation inherent to standard statistical regression. Models built using this methodology are optimized to the data with which they are constructed. Thus they will only be as representative as the data used in their creation are representative. The more data available, the more likely this will be the case. But still, any estimate provided by the model from input parameters that lie within the range of the data used for calibration will be an interpolation; worse still, estimates based on input parameters that lie outside the range of the calibration data will be an extrapolation. Regression models are generally considered to be much less trustworthy when used in this latter context.

In short, a regression-based model is only as good as the data that produced it. However, if the available empirical data is limited, by invoking Bayes’ theorem (see section 2.2.8), the collective “anecdotal” experience of individuals well-versed in the
The domain of interest can be used to figuratively expand the range and improve the "representativeness" of the data available for use.

Table 3.1 - Predictive Accuracy of Three Different Calibration Approaches for COCOMO II Using the Same 161 Data Points (Unstratified)

<table>
<thead>
<tr>
<th>Prediction Accuracy Level</th>
<th>Delphi-based Expert Judgement</th>
<th>10% Weighted Average-based</th>
<th>Bayesian-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRED (.20)</td>
<td>48%</td>
<td>52%</td>
<td>63%</td>
</tr>
<tr>
<td>PRED (.25)</td>
<td>55%</td>
<td>61%</td>
<td>68%</td>
</tr>
<tr>
<td>PRED (.30)</td>
<td>61%</td>
<td>68%</td>
<td>75%</td>
</tr>
</tbody>
</table>

Table 3.1 show the improvements in model fidelity Devnani-Chulani
demonstrated using a Bayesian approach to calibrate COCOMO II compared to calibrations based first solely on expert judgement and then using standard regression techniques coupled with a 10% weighted averaging of expert opinion with empirical data.*

The Bayesian approach combines the objective empirical and subjective expert information via the following general formula:4

\[
m_{a-posteriori} = \sigma^2_{a-posteriori} \cdot \left[ \left( \frac{n}{\sigma^2_{data}} \right) m_{data} + \left( \frac{1}{\sigma^2_{a-priori}} \right) m_{a-priori} \right]
\]

Eq. 3.1a

* COCOMO II.1997 was calibrated on 83 data points using standard regression techniques coupled with a 10% weighted averaging of expert opinion with empirical data. In other words, the numerical value of any given model parameter was based on a simple weighted sum equal to 90% of an expert-derived value added to 10% of an empirically-based linear regression-derived value. COCOMO II.2000 was calibrated on 161 data points using Bayesian techniques that combined expert-derived and linear regression-derived parameter values in a weighted average according to the variance of their respective distributions. Devnani-Chulani did an experiment, however, running the 10% weighted average technique over those same 161 data points, the results of which are shown in table 3.1 and which demonstrate that the improvements in model accuracy between the 1997 and 2000 calibrations of COCOMO II (see chapter 4 of chapter 1 endnote #11) were attributable primarily to the change to the Bayesian calibration technique as opposed to the increase in available calibration data.
where

\[ \sigma^2_{a\text{-posteriori}} = \left( \frac{n}{\sigma^2_{\text{data}}} + \frac{1}{\sigma^2_{a\text{-priori}}} \right)^{-1} \]  

Eq. 3.1b

and

\[ m_{a\text{-posteriori}} = \text{the Bayesian-derived posterior value of the given parameter.} \]
\[ m_{\text{data}} = \text{the mean value of the given parameter as determined from sample data.} \]
\[ m_{a\text{-priori}} = \text{the prior intuitive value of the given parameter.} \]
\[ \sigma^2_{a\text{-posteriori}} = \text{the variance of the posterior value of the given parameter.} \]
\[ n = \text{the number of observations in the sample data.} \]
\[ \sigma^2_{\text{data}} = \text{the variance of the distribution of the sample data.} \]
\[ \frac{n}{\sigma^2_{\text{data}}} = \text{the variance of the mean value of the given parameter as determined from sample data.} \]
\[ \frac{n}{\sigma^2_{\text{data}}} = \text{the precision of the mean value of the given parameter as determined from sample data.} \]
\[ \sigma^2_{a\text{-priori}} = \text{the variance of the intuitive prior value of the given parameter.} \]
\[ \frac{1}{\sigma^2_{a\text{-priori}}} = \text{the precision of the intuitive prior value of the given parameter.} \]

For COCOMO II, the specific parameters estimated via Bayesian analysis are a set of beta coefficients that appear in the log transform linear regression formulation of the basic COCOMO II modeling equations (see equations 3.3a through 3.3c, 3.4 and 3.5). The above general equations are thus recast below in matrix notation in terms of those regression coefficients:
\[
\beta_{a-posteriori} = (I_{a-posteriori})^{-1} \cdot (I_{data} \beta_{data} + I_{a-priori} \beta_{a-priori}) \\
= V_{a-posteriori} \cdot [(1/S^2)X'X\beta_{data} + H\beta_{a-priori}]
\]

where

\(\beta_{a-posteriori}\) = the matrix of the \textit{a-posteriori} beta coefficients.

\(\beta_{data}\) = the matrix of regression-derived beta coefficients.

\(\beta_{a-priori}\) = the matrix of \textit{a-priori} beta coefficients.

\(I_{a-posteriori}\) = the matrix of the precision of the \textit{a-posteriori} beta coefficients.

\((I_{a-posteriori})^{-1}\) = the inverse matrix of \(I_{a-posteriori}\).

\(= V_{a-posteriori}\) (the matrix of variance of the \textit{a-posteriori} beta coefficients).

\(= [(1/S^2)X'X + H]^{-1}\).

\(I_{data}\) = the matrix of the precision of the regression-derived beta coefficients.

\(= (1/S^2)X'X\).

\(I_{a-priori}\) = the matrix of the precision of the \textit{a-priori} beta coefficients.

\(= (V_{a-priori})^{-1}\) (the inverse matrix of the variance of the \textit{a-priori} beta coefficients).

\(= H\)

and

\(X\) = the matrix of sample observations for the predictor variables in the log transform version of the basic model.

\(X'\) = the transpose matrix of \(X\).

\(X'X\) = the matrix of the number of sample observations for each predictor variable.

Eq. 3.2a

Eq. 3.2b
\((1/S^2)\) = the matrix of the reciprocal of the variance of the distribution of the sample observations for each predictor variable.

From this formulation it can be seen that the posterior value of the beta coefficients represents a weighted average of the empirically-derived and expert-derived values of beta. The weighting in this case is based on the relative variances of the empirically and expert-based values, the greater weighting given to that item with the smaller variance. Recall, this was illustrated in figure 2.9 in section 2.2.8. Equation 3.1a is the mathematical basis for figure 2.9.

Not shown in figure 2.9, however, is the fact that the posterior variance of \(\beta_{a.posteriori}\) will always be smaller than the variances of either the empirically or expert-derived distributions by themselves. The definition given above for \((1/\beta_{a.posteriori})^{-1}\) indicates the reason this is true. It shows that the variance of \(\beta_{a.posteriori}\) is the inverse of the combined precision of the empirically and expert-derived information. The greater the precision, the smaller the variance, so the summing of these values together guarantees that the combined variance will be smaller than that of either individually.

As for how this technique is might be applied to COCOTS in the future, recall the statement in section 3.1 that the first pass COTS estimation model was developed using the methodology shown in figure 3.1, except that the calibration performed in step six was not Bayesian-based. The linear proportionality constant \(A_{COT}\) in equation 4.1a (see section 4.1.1) was determined via normal regression while the numerical values for the linear effort multipliers \(EM_{COT(i)}\) in that equation were wholly expert-
based. The same was true for the second pass model discussed in section 4.2.1. The constant $A_{\text{COT}}$ in equation 4.2a was determined via classical regression while the numerical values for all the remaining parameters remained expert-based.

Again, recall that between the second and third pass models, COCOTS was split from one model into four different submodels (assessment, tailoring, glue code, and volatility), and subsequently reduced to three submodels in the fourth pass model (assessment, tailoring, and glue code). Other than the glue code model, the new submodels don't really lend themselves to Bayesian techniques. The reason is that because of the way these models are formulated (see section 4.2.4), it is very difficult to generate expert-based judgments of their particular parameter values. Since no prior estimates can be determined, the Bayesian technique cannot be applied. The models in these cases will have to be solely empirically-based.

It is not that these other submodels are oddly constructed—in fact, mathematically they are quite simple. The germane parameters are based on simple averages. It is just that so little has been published as of this writing regarding the kind of information these parameters are trying to model that experts are unable to form their own opinions regarding their likely values. Thus, at this point one must rely only on reported numbers.

With the other submodels excluded, that leaves only the glue code submodel as a candidate for Bayesian-assisted calibration. Again, this model is similar in form to COCOMO II (compare equations 2.9a through 2.9b in section 2.4.1 to equations 4.5a
through 4.5b in section 4.2.1). As such, the same general procedures used by Devnani-Chulani to apply Bayesian techniques to COCOMO II would be used here.

3.2.5 Step 7: Model Refinement & Renewed Data Collection

This is the final step in the modeling methodology diagramed in figure 3.1. Having arrived at a fully realized estimation model at the completion of step six, this is now the stage at which that model is examined and tested in depth. In particular, the model is explored for insights that might be used to improve its overall estimation abilities.

Examination of the first pass model led to a return to step three in figure 3.1. The model was reworked, resulting in changes in parameter definitions and rating criteria, followed by the collection of new industrial data for use in calibration. Even with the parameter redefinitions, however, the general form of that second pass model remained the same as the earlier first pass model.

In depth examination of the second pass model finally led to the conclusion, however, that that existing basic form of the COTS estimation model was seriously deficient. The problem was that the model as formulated assumed that the bulk of the effort associated with using COTS components as part of a software system was to be found in the writing of code needed to bind the COTS components into that system. Thus, the more binding-ware or "glue code" that needed to be written, obviously the more effort that would be involved, just as with the new application or custom coding activity modeled by COCOMO II.
And this appeared to be true as far as it went. But concentrating only on the activity associated with glue code ignored the fact that when using COTS components, a lot of effort must be expended that has nothing to do with the amount of glue code that must be created. For instance, a lot of specialized work must be put into selecting which COTS components to use. In the end, after expending all that investigative effort, it is entirely possible that no COTS components will be deemed suitable for use. No COTS components means no glue code, and thus according to the first pass and second pass models, no COTS-related effort—which is clearly untrue in this case.

Realization of these kinds of deficiencies in the first pass and second pass models led to yet another return to step three in figure 4.1, this time resulting in a radical restructuring of the overall model in a third pass version. Within this latest version, the second pass model became a submodel still dedicated to glue code effort, but three other submodels were added to account for effort associated with COTS product assessment, tailoring, and volatility, respectively. In turn, this created a need to collect even more industrial data to accommodate calibration of the new submodels.

One more iteration on the third pass model from steps five through seven in figure 3.1 (excluding the Bayesian analysis) was performed to conclude this current research. This fourth iteration model differs from the third iteration models in two ways. First, the architectural scale factor AAREN has been redefined to correspond with the parameter from COCOMO II called RESL because it was realized they were
both attempting to capture the same information and it is unlikely there would be
differences in the ratings applied to these two factors within the same software
project. Second, the system volatility submodel has been eliminated. This submodel
was complicated, difficult to gather data for calibration, and unnecessarily blurred
the boundaries between COCOMO II and COCOTS because it was estimating effort
that would be incurred within the original software components of a CBS. Moreover,
the COCOMO II REVL parameter (see equation 2.9b) already can account for this
effort so it was concluded the submodel was redundant.
3.3 Chapter 3 Endnotes


4. Research Contributions

4.1 COCOTS Development Phase Effort Model Calibration

The model has gone through much evolution, culminating in the current form and calibration discussed at the end of this section. The earlier stages of the model are discussed below.

4.1.1 First Iteration: Initial COTS Integration Cost Model Study

Figure 4.1 is a conceptual representation of the first pass model resulting from the original COTS integration study and how it was viewed in relation to COCOMO.
II. Central is the notion that the COTS estimation model being developed with this research has always been thought of as an extension to COCOMO II rather than as a separate stand-alone model. The idea has been to create a complete CBS estimation capability within one tool that can model the use of new components, reuse components, and off-the-shelf components with equal ease.

With that in mind, the darker box on the bottom of figure 4.1 represents the CBS development effort that would be estimated by COCOMO II itself. This would cover the effort associated with newly developed items and reuse items. The lighter box on top represents the CBS development effort that would be estimated (in this case) by the first pass COTS estimation model. This would cover the effort associated with COTS items, which in the first pass model was thought to consist primarily of the work required to create the COTS glue code. To get the estimate for the entire effort required to develop a given COTS-based system, you would sum the COCOMO II and COTS model estimates together.

The items LCO, LCA, and IOC in the figure represent significant milestones in the standard software development life cycle. In particular note that COCOMO II only models activity after system requirements have already been established (the LCO milestone). Showing the COTS glue code development beginning after LCA, the preliminary design review milestone, is consistent with when the coding of new software elements begins under COCOMO II.
There are some important things to understand though about the software development life cycle when off-the-shelf components are added to the mix and which have affected the evolution of COCOTS.

COTS integration activities that facilitate the use of off-the-shelf components in a system design follow their own unique life cycle when compared to those activities associated with developing new components or adapting reuse components. The most significant difference perhaps is the additional step of pre-qualifying or assessing COTS components that gets added to the traditional software development cycle of determine requirements, design, code, integrate, test, and delivery that was illustrated way back in figure 1.1.

Figure 4.2 shows this assessment activity occurring prior to the main project development phase, but in fact this can occur during the opening stages of the project as well (with the subsequent iterations on the model it became clear this latter approach is actually more the norm). The key concept is that sometimes system requirements dictate which COTS components can feasibly be used, and sometimes it is the availability (or lack thereof) of certain COTS components that determines the final system requirements.
Figure 4.2 – COTS assessment activity in the CBS development life cycle.

This issue of when COTS assessment occurs is important because in the first pass model, the effort resulting from COTS assessment activity was presumed to lie outside the scope of the model. This was to keep the model inline with the activity phases covered by COCOMO II, which does not directly cover requirements specification activities. Upon completion of the first pass model, however, it became clear this was an unworkable assumption. Analysis of the results from the first past model showed that the original calibration data as reported likely contained some assessment effort, a result of the lack of precision of some data definitions. Moreover, the reality is that to reap the full benefit of using a COTS approach to constructing software systems, system requirements definition and COTS component selection must be tightly coupled, one trading off the other in an iterative fashion.

Mathematically, the first pass model took the following form:

\[
\text{Effort}_{COT} = A_{COT} (E_{size_{COT}})^{B_{COT}} \prod_{i=1}^{13} E_{M_{COT(i)}}
\]

Eq. 4.1a

where
\[ E \text{size}_{COT} = Size_{COT} \cdot (1 + \text{Brak}/100) \quad \text{Eq. 4.1b} \]

and

\[ \text{Effort}_{COT} = \text{glue code development effort in person-months.} \]

\[ E \text{size}_{COT} = \text{effective size of the glue code under development after adjusting} \]
\[ \text{for rework that must be done as a result of changes in} \]
\[ \text{requirements or volatility in the COTS products for which the} \]
\[ \text{glue code is being written.} \]

\[ Size_{COT} = \text{absolute size of the glue code under development in function points.} \]

\[ A_{COT} = \text{a multiplicative conversion constant relating glue code size to} \]
\[ \text{development effort, now representing the productivity that typically} \]
\[ \text{obtains when project conditions allow all thirteen multiplicative} \]
\[ \text{“effort multiplier” parameters EM}_{COT(i)} \text{ in the model to be assigned} \]
\[ \text{their baseline “nominal” ratings, thus reducing their collective impact} \]
\[ \text{to nil.} \]

\[ EM_{COT(i)} = \text{“effort multipliers” that either increase or decrease the nominal} \]
\[ \text{effort estimate given by the equation based upon characterizations} \]
\[ \text{of the environmental conditions that exist while the system is} \]
\[ \text{under development; their nominal value is 1.0.} \]

\[ B_{COT} = \text{an exponential factor that accounts for nonlinear economies or} \]
\[ \text{diseconomies of scale that may accrue as glue code increases in size;} \]
\[ \text{set } = 1 \text{ for modeling simplicity in the first pass model with the} \]
\[ \text{expectation that later versions of the model would offer opportunities} \]
\[ \text{to more precisely calibrate this parameter.} \]

\[ \text{Brak = estimated percentage of glue code “breakage” during development;} \]
\[ \text{this is code that must be reworked due to changes in requirements or} \]
\[ \text{release of an updated COTS product but explicitly not as a result of} \]
\[ \text{programmer error.} \]

There were thirteen multiplicative cost drivers EM_{COT(i)} defined for the first pass COTS model summarized in table 4.1 and then followed by their definitions.
Table 4.1 - First Pass Model Parameters

<table>
<thead>
<tr>
<th>Effort Multipliers – Personnel Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) CIEP – COTS Integrator Experience with Product</td>
</tr>
<tr>
<td>2) CIPC – COTS Integrator Personnel Capability</td>
</tr>
<tr>
<td>3) CIIX – COTS Integrator Experience with COTS Integration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Effort Multipliers – COTS Component Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>4) CPDM – COTS Product and Documentation Maturity</td>
</tr>
<tr>
<td>5) CVEW – COTS Vendor Product Extension Willingness</td>
</tr>
<tr>
<td>6) CREL – COTS Reliability</td>
</tr>
<tr>
<td>7) CCOS – COTS Compliance with Open Face Standards</td>
</tr>
<tr>
<td>8) CPER – COTS Performance</td>
</tr>
<tr>
<td>9) CVMS – COTS Vendor Maturity and Product Support</td>
</tr>
<tr>
<td>10) CVPT – COTS Vendor Provided Training</td>
</tr>
<tr>
<td>11) CPRT – COTS Portability</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Effort Multipliers – Application/System Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>12) CPAX – COTS Product and Application Complexity</td>
</tr>
<tr>
<td>13) CIAR – COTS Integrator Architecture/Risk Resolution</td>
</tr>
</tbody>
</table>

The definitions for these drivers are as follows:

**Personnel Drivers**

- **COTS Integrator Experience with Product (CIEP):** How much experience does the development staff have with running, integrating, and maintaining the COTS product?

---

*This term is similar to the REVL term in COCOMO II as described in section 2.4. The difference is that now code rework due to volatility in the COTS products is also added to the definition.*
• **COTS Integrator Personnel Capability (CIPC):** What are the overall software development skills and abilities that your personnel bring to the COTS product integration task?

• **COTS Integrator Experience with COTS Integration (CIXI):** How much experience does the development staff have with assessing, integrating, and adapting to upgrades of COTS products in general?

* **COTS Component Drivers**

• **COTS Product and Documentation Maturity (CPDM):** How many copies of the COTS product have been sold? How long has it been on the market? Has the product established a reputation for utility and reliability, i.e., a known track record? Does the product come with the necessary, well-written documentation to install, maintain, and use the package?

• **COTS Vendor Product Extension Willingness (CVEW):** How willing is the vendor of the COTS product to modify the design of their software to meet your specific needs, either by adding or removing functionality or by changing the way it operates?

• **COTS Reliability (CREL):** Does the COTS product meet or exceed the same standards of reliability as is required of the system as a whole into which the product is being integrated?
• **COTS Compliance with Open Interface Standards (CCOS):** How well does the COTS product comply with accepted industry external and internal interface standards?

• **COTS Performance (CPER):** How well does the COTS product meet or exceed the same standards of performance as is required of the system as a whole into which the product is being integrated?

• **COTS Vendor Maturity and Product Support (CVMS):** How long has the vendor been in business? Are they a known quantity, or are they a new start-up? Have their products established a reputation for reliability? Even if they have been in business for awhile, how well do they provide technical support for their products (either directly or through third parties)?

• **COTS Vendor Provided Training (CVPT):** How much training will the vendor provide (either directly or through third parties)?

• **COTS Portability (CPRT):** How well does the COTS product meet or exceed the same standards of portability as is required of the system as a whole into which the product is being integrated?

**Application/System Drivers**

• **COTS Product and Application Complexity (CPAX):** What kind of system are you building? Pushing software technology to state-of-the-art? Real time transaction monitoring, or basic file maintenance? Are there difficult
synchronization issues? Does the system have to balance conflicting criteria (e.g., security, safety, accuracy, ease of use, speed)?

- **COTS Integrator Architecture/Risk Resolution (CIAR):** How much effort is expended by your integration staff in ensuring that potential risks to the COTS integration task are identified and mitigated, including through the examination of potential architectural mismatches between the COTS components and the overall system, or between the COTS components themselves? How thorough is the project’s Software Architecture Review?

Like the effort multipliers for COCOMO II, each of the above drivers has set of criteria associated with it for rating on a five point scale from Very Low to Very High. (The details of these can be found in chapter 1 endnote #18).

As discussed in section 3.2.5, the first pass model was calibrated to a set of six carefully controlled graduate student software design projects that included the use of COTS components. Still, the overall accuracy of this initial model was low; at best, only 33% of the projects were estimated to within 40% of actual reported effort. The main reason was found to be the preponderance of effort going into COTS assessment for at least two of the projects that was not being reflected in the corresponding size of the glue code being developed for these projects. The goals of the calibration on the student projects were realized, however, in that when compared to the results obtained from the industrial data that was available at the time (again, see section 3.2.5), the calibration based on the student projects was
superior. This demonstrated an ability to improve the accuracy of the COTS estimation model if the correct kind of data were to be available.

4.1.2 Second Iteration: Initial FAA Revision

Figure 4.3 illustrates the second pass model, which was also the first to adopt the name COCOTS to underscore its connection to COCOMO II. Though retaining the same general form as the first pass model, this revision undertook a wholesale redefinition of the model parameters, including the addition of one more linear effort multiplier and the promotion of the linear multiplier CIAR to a nonlinear scale factor called AAREN. The intent was to try to tighten up the parameter descriptions in such a way as to reduce as much as possible overlap in the concepts being captured by each parameter. This of course also required a complete revision of the rating scales for each parameter. The model also tried to acknowledge that there was more effort being captured in the reported historical data than just that associated directly with the creation of glue code. This is represented in figure 4.3 by the increase in relative size of the COTS model block compared to its representation in figure 4.1. Notice that the start of glue code development is now presumed to begin sometime between the LCO and LCA milestones—in other words, well before overall system design details are finalized.
Mathematically, the second pass model took this form:

\[ \text{Effort}_{\text{COT}} = A_{\text{COT}} \left( \text{Esize}_{\text{COT}} \right)^{E_{\text{COT}}} \prod_{i=1}^{14} EM_{\text{COT}(i)} \]  

Eq. 4.2a

where

\[ \text{Esize}_{\text{COT}} = \text{Size}_{\text{COT}} \cdot (1 + \text{Brak}/100) \]  

Eq. 4.2b

and

\[ E_{\text{COT}} = B_{\text{COT}} + (0.04 \cdot SF_{\text{COT}}) \]  

Eq. 4.2c

and

\[ \text{Effort}_{\text{COT}} = \text{glue code development effort in person-months}. \]

\[ \text{Esize}_{\text{COT}} = \text{effective size of the glue code under development after adjusting for rework that must be done as a result of changes in} \]
requirements or volatility in the COTS products for which the glue code is being written.

\[ \text{Size}_{\text{COT}} = \text{absolute size of the glue code under development in source lines of code.} \]

\[ \text{A}_{\text{COT}} = \text{a multiplicative conversion constant relating glue code size to development effort, now representing the productivity that typically obtains when project conditions allow all fourteen multiplicative \text{"effort multiplier" parameters EM}_{\text{COT(i)}} \text{ in the model to be assigned their baseline \text{"nominal" ratings, thus reducing their collective impact to nil.}} \]

\[ \text{EM}_{\text{COT(i)}} = \text{\text{"effort multipliers" that either increase or decrease the nominal effort estimate given by the equation based upon characterizations of the environmental conditions that exist while the system is under development; their nominal value is 1.0.}} \]

\[ \text{Brak} = \text{estimated percentage of glue code \text{"breakage" during development; this is code that must be reworked due to changes in requirements or release of an updated COTS product but explicitly not as a result of programmer error.}} \]

\[ \text{E}_{\text{COT}} = \text{an exponential factor that accounts for nonlinear economies or diseconomies of scale that may accrue as glue code increases in size and which in turn is now a function of a constant B}_{\text{COT}} \text{ and a single scale factor SF}_{\text{COT}}. \]

\[ \text{B}_{\text{COT}} = \text{a constant appearing in the exponential factor that represents the costs or savings that still obtain even when project conditions allow the absolute best possible ratings to be assigned to the scale factor SF}_{\text{COT}}, \text{ reducing its impact to nil; until data is available to more precisely calibrate this parameter, set = 1 in the second pass model to reflect the commonly held wisdom that even under the best possible system-wide conditions, \text{economies} of scale will not become evident as glue code increases in size.} \]

\[ \text{SF}_{\text{COT}} = \text{a \text{"scale factor" characterizing project conditions that have been shown to have nonlinear impacts on software development effort determining whether economies or diseconomies of scale will likely present during the development.}} \]
The fourteen multiplicative cost drivers $EM_{COT(i)}$ and one scale factor $SF_{COT}$ defined for the second pass COTS model are summarized in table 4.2, again followed by definitions.

<table>
<thead>
<tr>
<th>Table 4.2 – Second Pass Model Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nonlinear Scale Factor</strong></td>
</tr>
<tr>
<td>AAREN – Application Architectural Engineering</td>
</tr>
<tr>
<td><strong>Effort Multipliers – Personnel Drivers</strong></td>
</tr>
<tr>
<td>1) ACIEP – COTS Integrator Experience with Product</td>
</tr>
<tr>
<td>2) ACIPC – COTS Integrator Personnel Capability</td>
</tr>
<tr>
<td>3) AXCIP – Integrator Experience with COTS Integration Processes</td>
</tr>
<tr>
<td>4) APCON – Integrator Personnel Continuity</td>
</tr>
<tr>
<td><strong>Effort Multipliers – COTS Component Drivers</strong></td>
</tr>
<tr>
<td>5) ACPMT – COTS Product Maturity</td>
</tr>
<tr>
<td>6) ACSEW – COTS Supplier Product Extension Willingness</td>
</tr>
<tr>
<td>7) APCPX – COTS Product Interface Complexity</td>
</tr>
<tr>
<td>8) ACPPS – COTS Supplier Product Support</td>
</tr>
<tr>
<td>9) ACPTD – COTS Supplier Provided Training and Documentation</td>
</tr>
<tr>
<td>10) APVOL – COTS Product Volatility</td>
</tr>
<tr>
<td><strong>Effort Multipliers – Application/System Drivers</strong></td>
</tr>
<tr>
<td>11) ACREL – Constraints on Application System/Subsystem Reliability</td>
</tr>
<tr>
<td>12) AACPX – Application Interface Complexity</td>
</tr>
<tr>
<td>13) ACPER – Constraints on COTS Technical Performance</td>
</tr>
<tr>
<td>14) ASPRT – Application System Portability</td>
</tr>
</tbody>
</table>
The definitions for these drivers are as follows:

Exponential scale factor $SF_{COT}$:

- **Application Architectural Engineering (AAREN):** How adequate/sophisticated were the techniques used to define and validate the overall systems architecture?

Effort multipliers $EM_{COT(i)}$:

**Personnel Drivers**

- **COTS/NDI Integrator Experience with Product (ACIEP):** How much experience did/does the development staff have with running, integrating, and maintaining the COTS/NDI products?

- **COTS/NDI Integrator Personnel Capability (ACIPC):** What were/are the overall software development skills and abilities which your team as a whole on average brought/bring to the product integration tasks AS WELL AS experience with the specific tools, languages, platforms, and operating systems used/being used in the integration tasks?

- **Integrator Experience with COTS/NDI Integration Processes (AXCIP):** Does a formal and validated COTS/NDI integration process exist within your organization and how experienced was/is the development staff in that formal process?
- **Integrator Personnel Continuity (APCON):** How stable was/is your integration team? Are the same people staying around for the duration of the tasks, or must you keep bringing in new people and familiarizing them with the particulars of the project because experienced personnel leave?

**COTS Component Drivers**

- **COTS/NDI Product Maturity (ACPMT):** How many copies have been sold or used previously of the major versions (as opposed to release of those versions) of the COTS/NDI components you integrated or intend to integrate? How long have the versions been on the market or available for use? How large are the versions’ market shares or installed user bases? How thoroughly have the versions been used by others in the manner you used or intend to use them?

- **COTS/NDI Supplier Product Extension Willingness (ACSEW):** How willing were/are the suppliers of the COTS/NDI products to modify the design of their software to meet your specific needs, either by adding or removing functionality or by changing the way it operates? In the case of COTS components, this refers to changes that would appear in market releases of the product. In the case of NDI components, this refers to changes that would appear in copies being distributed to all users of the component. This does
NOT include specialty changes in the COTS/NDI component that would appear in your copy only.

- **COTS/NDI Product Interface Complexity (APCPX):** What are the nature of the interfaces between the COTS/NDI components and the glue code connecting them to the main application? Are there difficult synchronization issues? Must the interfaces balance conflicting criteria (e.g., security, safety, accuracy, ease of use, speed)?

- **COTS/NDI Supplier Product Support (ACPPS):** What is the nature of the technical support for the COTS/NDI components that was/is available AND PROCURED for the integration team during the development, either directly from the component suppliers or through third parties?

- **COTS/NDI Supplier Provided Training and Documentation (ACPTD):** How much training and/or documentation for the COTS/NDI components was/is available AND PROCURED for the integration team during the development, either directly from the component suppliers or through third parties?

- **COTS Product Volatility (APVOL):** How many releases of the COTS component (or patches to releases) were/can be expected to be issued by the component supplier during the development?
**Application/System Drivers**

- **Constraints on System/Subsystem Reliability (ACREL):** How severe are the overall reliability constraints on the system or subsystem into which the COTS/NDI components was/is being integrated? What are the potential consequences if the components fail to perform as required in any given time frame? (Note that availability is considered an issue different than reliability and is NOT addressed in this cost driver.)

- **Application Interface Complexity (AACPX):** What are the nature of the interfaces between the main application system or subsystem and the glue code used to connect the system to the COTS/NDI components? Are there difficult synchronization issues? Must the interface balance conflicting criteria (e.g., security, safety, accuracy, ease of use, speed)?

- **Constraints on System/subsystem Technical Performance (ACPER):** How severe were/are the technical performance constraints (e.g., storage, memory, reserve, flow through capacity, etc.) on the application system or subsystem that the COTS/NDI components needed to/must meet?

- **System Portability (ASPR):** What were/are the overall system or subsystem portability requirements that the COTS/NDI component needed to/must meet?
The details of the five point rating scale criteria for the above drivers can be found in chapter 1 endnote #20.

The second pass model was calibrated to a set of five industrial project data points gathered with the assistance of the FAA. The results of this calibration were only marginally better than those achieved for the first pass model. But they underscored even more that there was a fundamental problem with how the model was formulated. The main issue again was that the model presumed COTS associated effort in a CBS development is essentially proportional to the amount of glue code that must be written. Analysis of the model results indicated that clearly this is not the case. For example, one of the historical CBS projects used to calibrate the model reported significant COTS-related effort but zero lines of glue code. As formulated, the second pass model would thus estimate the COTS-related effort also as zero, which was wrong. This lead to a radical rethinking of the overall structure of COCOTS, resulting in the third pass model.

4.1.3 Third Iteration: Second FAA/ONR Revision

As illustrated in figure 4.4, the third pass model represents a significant restructuring of the COTS estimation model. COCOTS has now been decomposed from one into four related submodels, each addressing individually what have been identified from study of the two previous models as the four primary sources of COTS software integration costs.
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, and (4) increased system level programming and testing due to volatility in incorporated COTS components.

Assessment is the process by which COTS components are evaluated and 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 component for use regardless of the system into which it is being incorporated—in other words
the means whereby COTS software products are configured for use in a specific context. These are things such as initializing parameter values, specifying Input/Output 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.

As for life cycle, as of now COCOTS still addresses only initial development costs associated with using COTS software components. However, as a result of splitting COTS assessment activities into their own submodel, requirements definition is now explicitly recognized as an activity being covered by the model, since it is now recognized that it is impossible to separate COTS selection from requirements definition. This is a significant change. Requirements definition was not included in the original model because this activity traditionally has not been covered by COCOMO estimates. Including it in the COCOTS model now puts that model out of alignment with COCOMO II, indicated in figure 4.4 by the offset of the assessment model block relative to the central COCOMO II block. This misalignment will have to be reconciled before the individual effort estimates from
COCOMO and COCOTS can be combined into a single schedule estimating function.

The mathematical form for each the submodels is different. A common feature, however, is that estimates are done based upon the classes of COTS components being examined. This is another significant change from the previous versions of the COTS estimation model. It was a result of insights gained during some of the data collection interviews that have been conducted to date. Grouping COTS products by basic function such as GUI builders, operating systems, databases, word processors, etc., has been found to be the most effective way to gather industrial calibration data. Managers and other data providers have found it difficult to parse COTS integration data at either the individual component or system level, but thinking in terms of classes of components has proven to be a fairly straightforward way for them to ferret out the numbers about which they are being asked.

**Assessment**

The assessment submodel is formulated as follows:

\[ \text{Effort}_A = \text{IFE} + \text{DAE} \]  \hspace{1cm} \text{Eq. 4.3a}

where

\[ \text{IFE} = \sum_{i=1}^{m} (\text{CCF}_i \cdot \text{MIFE}_i) \]  \hspace{1cm} \text{Eq. 4.3b}

and

\[ \text{DAE} = \sum_{j=1}^{n} (\text{CCD}_j \cdot \text{MDAE}_j) \]  \hspace{1cm} \text{Eq. 4.3c}

and
Effort\textsubscript{A} = total COTS product assessment effort for the project.

IFE = total initial filtering effort.

DAE = total detailed assessment effort.

M = number of different classes of COTS products going through initial filtering.

N = number of different classes of COTS products going through detailed assessment.

CCF\textsubscript{i} = number of candidate COTS products within a given class of products going through initial filtering.

CCD\textsubscript{j} = number of candidate COTS products within a given class of products going through detailed assessment.

MIFE\textsubscript{i} = mean initial filtering effort for a given class of COTS products.*

\* I went back and forth as to whether it was better to use the mean or the median value of the sample data for the calibrated parameters required in equations 5.3b and 5.3c (and for similar parameters used in the other submodels in the third pass model). With the limited and sometimes skewed historical project data currently in hand, it was difficult to decide which estimator would offer the most "representative" parameter. I finally resolved the issue by going back to what are considered desirable properties of statistical point estimators. Three of the most prominent of these properties are sufficiency, unbiasedness, and efficiency.

A sufficient statistic is one that captures all the information in a sample that is relevant with respect to the uncertainty surrounding the item that is being estimated. For example, suppose you work at a music marketing firm, and your boss tells you to find out the average age of 1,000 people attending a concert in a large auditorium. You don't have the time to ask all of those people individually their ages, so instead you assume that they represent a normal population, randomly select 100 of them and ask everyone in that smaller group their ages. You then calculate the mean age of that smaller group and use that value as an estimate for the mean age of all 1,000 people in the auditorium. The mean of the smaller group is considered a sufficient statistic because giving your boss a list containing the ages of all 100 people in that smaller group will not provide him anymore information about the average age of everyone at the concert than just telling him the average age of the smaller group. (The complete list might give him more insight into the median or the most typical (mode) age of people at the concert, but not the average age.)

An unbiased statistic is one whose expected value is equal to the actual value of the item being estimated. Returning to our example, the mean of the smaller group is considered an unbiased statistic because its expected value is the same as the actual average age of all 1,000 people at the concert.

An efficient statistic is one that has as great a precision as possible. In other words, an estimator with a small variance is more desirable than a similar estimator with a larger variance, all other things being equal. Since the mean and the median of a perfectly normal distribution are the same, it might be
MDAE_j = mean detailed assessment effort for a given class of COTS products.

During the first two iterations on the COTS estimation model, a few insights were realized on the COTS assessment process. First, selection of COTS products is typically based on an assessment of each product in light of certain product attributes. 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. Table 4.3 contains a set of assessment attributes identified that seem to cover most cases. (Expanded definitions for these attributes can be found in appendix B.)

Another point that came to light is that COTS assessment most often occurs in two stages. The first stage is usually just a quick evaluation of the market space for products that seem to warrant further detailed investigation. The goal at this point is to eliminate from further consideration products that clearly will not meet the objectives of the software system being developed. This activity is captured by equation 4.3b. The second stage then becomes a detailed evaluation of the remaining...
candidate COTS products, the goal now being to verify as thoroughly as possible whether or not a given COTS product will meet the needs of the system. This activity is captured by equation 4.3c.

<table>
<thead>
<tr>
<th>Potential COTS Product Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correctness</td>
</tr>
<tr>
<td>Availability/Robustness</td>
</tr>
<tr>
<td>Security</td>
</tr>
<tr>
<td>Product Performance</td>
</tr>
<tr>
<td>Understandability</td>
</tr>
<tr>
<td>Ease of Use</td>
</tr>
<tr>
<td>Version Compatibility</td>
</tr>
<tr>
<td>Intercomponent Compatibility</td>
</tr>
<tr>
<td>Flexibility</td>
</tr>
<tr>
<td>Installation/Upgrade Ease</td>
</tr>
<tr>
<td>Portability</td>
</tr>
<tr>
<td>Functionality</td>
</tr>
<tr>
<td>Price</td>
</tr>
<tr>
<td>Maturity</td>
</tr>
<tr>
<td>Vendor Support</td>
</tr>
<tr>
<td>Training</td>
</tr>
<tr>
<td>Vendor Concessions</td>
</tr>
</tbody>
</table>

The total COTS product assessment activity on a project then becomes the combination of the two kinds of effort, represented by equation 4.3a.

* These attributes were identified with the assistance of Dr. Clark (see footnote page 26) and were adapted from chapter 4 endnote #1, IEEE Standards Collection Software Engineering.
The tailoring submodel takes on this mathematical form:

$$\text{Effort}_T = \sum_{i=1}^{m} (NCT_i \cdot MTE_i \cdot TCQ_i) \quad \text{Eq. 4.4}$$

where

- $\text{Effort}_T$ = total COTS product tailoring effort for the project.
- $M$ = number of different classes of COTS products being tailored.
- $NCT_i$ = number of COTS products within a given class of products being tailored.
- $MTE_i$ = mean tailoring effort for a given class of COTS products.
- $TCQ_i$ = tailoring complexity qualifier; a linear multiplier that either increases or decreases the nominal tailoring effort for the given class of products based upon characterizations of the complexity of the tailoring required (the nominal value is 1.0).

The formulation of the tailoring submodel presumes that the expected difficulty of the tailoring work that is going to be required to get a given COTS product integrated into a system can be characterized by standardized rating criteria, represented by the parameter $TCQ$ in equation 4.4. Like the parameters in COCOMO II, this parameter is rated on a five point scale, the details of which can be found in appendix B.

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.
Glue Code

The glue code submodel takes on this mathematical form:

\[
\text{Effort}_{\text{GC}} = A_{\text{GC}} \, (\text{Esize}_{\text{GC}})^{E_{\text{GC}}} \prod_{i=1}^{13} \text{EM}_{\text{GC}(i)}
\]

Eq. 4.5a

where

\[
\text{Esize}_{\text{GC}} = \text{Size}_{\text{GC}} \cdot (1 + \text{CREVL}/100)
\]

Eq. 4.5b

and

\[
\text{E}_{\text{GC}} = B_{\text{GC}} + (0.04 \cdot \text{SF}_{\text{GC}})
\]

Eq. 4.5c

and

\[
\text{Effort}_{\text{GC}} = \text{glue code development effort in person-months.}
\]

\[
\text{Esize}_{\text{GC}} = \text{effective size of the glue code under development after adjusting for rework that must be done as a result of changes in requirements or volatility in the COTS products for which the glue code is being written.}
\]

\[
\text{Size}_{\text{GC}} = \text{absolute size of the glue code under development in source lines of code.}
\]

\[
A_{\text{GC}} = \text{a multiplicative conversion constant relating glue code size to development effort, now representing the productivity that typically obtains when project conditions allow all thirteen "effort multiplier" parameters EM}_{\text{GC}(i)} \text{ in the model to be assigned their baseline "nominal" ratings, thus reducing their collective impact to nil.}
\]

\[
\text{EM}_{\text{GC}(i)} = \text{"effort multipliers" that either increase or decrease the nominal effort estimate given by the equation based upon characterizations of the environmental conditions that exist while the system is under development (nominal = 1.0).}
\]

\[
\text{CREVL} = \text{estimated percentage of glue code that must be reworked during}
\]
development due to changes or evolution in requirements or release of an updated COTS product but explicitly not as a result of programmer error.*

\( E_{GC} \) = an exponential factor that accounts for nonlinear economies or diseconomies of scale that may accrue as glue code increases in size and which in turn is now a function of a constant \( B_{GC} \) and a single scale factor \( S_{FGC} \)

\( B_{GC} \) = a constant appearing in the exponential factor that represents the costs or savings that still obtain even when project conditions allow the absolute best possible ratings to be assigned to the scale factor \( S_{FGC} \) reducing its impact to nil; until data is available to more precisely calibrate this parameter, set \( = 1 \) in the third pass glue code submodel to reflect the commonly held wisdom that even under the best possible system-wide conditions, economies of scale will not become evident as glue code increases in size.

\( S_{FGC} \) = a "scale factor" characterizing project conditions that have been shown to have nonlinear impacts on software development effort determining whether economies or diseconomies of scale will likely present during the development.

The thirteen multiplicative cost drivers \( EM_{GC(i)} \) and one scale factor \( S_{FGC} \) defined for the third pass glue code submodel are summarized in table 4.4, again followed by definitions (these are the same as the second pass model but are repeated here for continuity).

* This is equivalent to the BRAK term in the second pass model. It has been renamed here to clarify its similarity to the REVL term in COCOMO II.
Table 4.4 – Third Pass Glue Code Submodel Parameters

<table>
<thead>
<tr>
<th>Exponential Scale Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAREN - Application Architectural Engineering</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Effort Multipliers - Personnel Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) ACIEP - COTS Integrator Experience with Product</td>
</tr>
<tr>
<td>2) ACIPC - COTS Integrator Personnel Capability</td>
</tr>
<tr>
<td>3) AXCIP - Integrator Experience with COTS Integration Processes</td>
</tr>
<tr>
<td>4) APCON - Integrator Personnel Continuity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Effort Multipliers - COTS Component Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>5) ACPMT - COTS Product Maturity</td>
</tr>
<tr>
<td>6) ACSEW - COTS Supplier Product Extension Willingness</td>
</tr>
<tr>
<td>7) APCPX - COTS Product Interface Complexity</td>
</tr>
<tr>
<td>8) ACPPS - COTS Supplier Product Support</td>
</tr>
<tr>
<td>9) ACPTD - COTS Supplier Provided Training and Documentation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Effort Multipliers - Application/System Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>10) ACREL - Constraints on Application System/Subsystem Reliability</td>
</tr>
<tr>
<td>11) AACPX - Application Interface Complexity</td>
</tr>
<tr>
<td>12) ACPER - Constraints on COTS Technical Performance</td>
</tr>
<tr>
<td>13) ASPRT - Application System Portability</td>
</tr>
</tbody>
</table>

The definitions for these drivers are as follows:

Exponential scale factor $SF_{EC}$:

- **Application Architectural Engineering (AAREN):** How adequate/sophisticated were the techniques used to define and validate the overall systems architecture?
Effort multipliers $\text{EM}_\text{GC(0)}$: 

**Personnel Drivers**

- **COTS/NDI Integrator Experience with Product (ACIEP):** How much experience did/does the development staff have with running, integrating, and maintaining the COTS/NDI products?

- **COTS/NDI Integrator Personnel Capability (ACIPC):** What were/are the overall software development skills and abilities which your team as a whole on average brought/bring to the product integration tasks AS WELL AS experience with the specific tools, languages, platforms, and operating systems used/being used in the integration tasks?

- **Integrator Experience with COTS/NDI Integration Processes (AXCIP):** Does a formal and validated COTS/NDI integration process exist within your organization and how experienced was/is the development staff in that formal process?

- **Integrator Personnel Continuity (APCON):** How stable was/is your integration team? Are the same people staying around for the duration of the tasks, or must you keep bringing in new people and familiarizing them with the particulars of the project because experienced personnel leave?
**COTS Component Drivers**

- **COTS/NDI Product Maturity (ACPMT):** How many copies have been sold or used previously of the major versions (as opposed to release of those versions) of the COTS/NDI components you integrated or intend to integrate? How long have the versions been on the market or available for use? How large are the versions' market shares or installed user bases? How thoroughly have the versions been used by others in the manner you used or intend to use them?

- **COTS/NDI Supplier Product Extension Willingness (ACSEW):** How willing were/are the suppliers of the COTS/NDI products to modify the design of their software to meet your specific needs, either by adding or removing functionality or by changing the way it operates? In the case of COTS components, this refers to changes that would appear in market releases of the product. In the case of NDI components, this refers to changes that would appear in copies being distributed to all users of the component. This does NOT include specialty changes in the COTS/NDI component that would appear in your copy only.

- **COTS/NDI Product Interface Complexity (APCPX):** What are the nature of the interfaces between the COTS/NDI components and the glue code connecting them to the main application? Are there difficult synchronization
issues? Must the interfaces balance conflicting criteria (e.g., security, safety, accuracy, ease of use, speed)?

- **COTS/NDI Supplier Product Support (ACPPS):** What is the nature of the technical support for the COTS/NDI components that was/is available AND PROCURED for the integration team during the development, either directly from the component suppliers or through third parties?

- **COTS/NDI Supplier Provided Training and Documentation (ACPTD):** How much training and/or documentation for the COTS/NDI components was/is available AND PROCURED for the integration team during the development, either directly from the component suppliers or through third parties?

**Application/System Drivers**

- **Constraints on System/Subsystem Reliability (ACREL):** How severe are the overall reliability constraints on the system or subsystem into which the COTS/NDI components was/is being integrated? What are the potential consequences if the components fail to perform as required in any given time frame? (Note that availability is considered an issue different than reliability and is NOT addressed in this cost driver.)

- **Application Interface Complexity (AACPX):** What are the nature of the interfaces between the main application system or subsystem and the glue
code used to connect the system to the COTS/NDI components? Are there difficult synchronization issues? Must the interface balance conflicting criteria (e.g., security, safety, accuracy, ease of use, speed)?

- **Constraints on System/subsystem Technical Performance (ACPER):** How severe were/are the technical performance constraints (e.g., storage, memory, reserve, flow through capacity, etc.) on the application system or subsystem that the COTS/NDI components needed to/must meet?

- **System Portability (ASPRT):** What were/are the overall system or subsystem portability requirements that the COTS/NDI component needed to/must meet?

The details of the five point rating scale criteria for the above drivers can be found in appendix B.

As indicated previously, the third pass glue code submodel is essentially the same in form as the second pass COTS estimation model. The only difference is the deletion of the linear effort multiplier APVOL. This term addressed the impact of volatility of a COTS product on the glue code that needs to be written. As this same concept is also captured in the term CREVL, it was decided APVOL was redundant.

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 $A_{GC}$, the rated nonlinear architectural scale factor $S_{FGC}$ and the individual rated effort multipliers $EM_{GC(i)}$. 

Finally, at this point a more explicit definition of glue code is in order. It 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 COCOTS, the following three-part definition was adopted:

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) code 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.

System Volatility

The system volatility submodel begins with one basic form as shown in equation 4.6. whereby the effort to rework the custom code in a CBS due to volatility in its COTS products is determined as a product of the original effort to create the custom
code and an "adjusted volatility proportionality factor." However, currently there are three variants proposed on how to determine that proportionality factor, each offering what is thought to be an increasingly more precise model:

\[ \text{Effort}_{SV} = \text{AVPF} \cdot \text{Effort}_{COC} \]  \hspace{1cm} \text{Eq. 4.6}

**Version 1**

\[ \text{AVPF} = (\text{NPVF}) \cdot \prod_{i=1}^{13} \text{EM}_{GC(i)} \]  \hspace{1cm} \text{Eq. 4.7}

**Version 2**

\[ \text{AVPF} = (\text{NVPF})^{E_{COC}} \cdot \prod_{i=1}^{13} \text{EM}_{GC(i)} \]  \hspace{1cm} \text{Eq. 4.8}

where for 1 & 2

\[ \text{NVPF} = (\text{SCREVL}/100) \]  \hspace{1cm} \text{Eq. 4.9}

\[ = \text{nominal estimated percentage (in decimal) of absolute custom code that must be reworked due to COTS product volatility where "absolute custom code" refers to the actual logical count of custom SLOC that will be delivered.} \]

and where for 2 & 3

\[ E_{COC} = B_{COC} + 0.01 \sum_{j=1}^{5} SF_{COC(j)} \]  \hspace{1cm} \text{Eq. 4.10}

**Version 3**

\[ \text{AVPF} = [(\text{NVPF} + 1)^{E_{COC}} - 1] \cdot \prod_{i=1}^{13} \text{EM}_{GC(i)} \]  \hspace{1cm} \text{Eq. 4.11}

where

\[ \text{NVPF} = (\text{SCREVL}/100)/(1+\text{REVFL}/100) \]  \hspace{1cm} \text{Eq. 4.12}

\[ = \text{nominal estimated percentage (in decimal) of effective custom code that is proportional to the nominal estimated percentage of absolute custom code that must be reworked due to COTS product volatility where "effective custom code" refers to the effective logical count} \]
of custom SLOC that must be developed accounting for changes in requirements in order to arrive at the absolute logical count of custom SLOC that will be delivered.

For all versions 1, 2 & 3 the following definitions hold:

\[ \text{Effort}_{sv} = \text{effort attributed to the custom code reworked due to COTS product volatility in person-months.} \]

\[ \text{Effort}_{coc} = \text{custom code development effort excluding effort due to COTS product volatility in person-months (see equation 2.9a).} \]

\[ \text{AVPF} = \text{adjusted volatility proportionality factor.} \]

\[ \text{NVPF} = \text{nominal volatility proportionality factor.} \]

\[ \text{EM}_{G_C(i)} = \text{the glue code "effort multipliers" that either increase or decrease the nominal effort estimate given by the equation based upon characterizations of the environmental conditions that exist while the system is under development (nominal = 1.0).} \]

\[ \text{SCREVL} = \text{nominal estimated percentage of absolute custom code that must be reworked due to COTS product volatility.} \]

\[ \text{REVL} = \text{estimated percentage of absolute custom code that must be reworked due to requirements evolution.} \]

\[ \text{E}_{coc} = \text{an exponential factor that accounts for nonlinear economies or diseconomies of scale that may accrue as software increases in size and which in turn is a function of a constant B}_{coc} \text{ and five "scale factors" SF}_{coc(j)} \text{(see equation 2.9b).} \]

\[ \text{B}_{coc} = \text{a constant appearing in the exponential factor that represents the costs or savings that still obtain even when project conditions allow the absolute best possible ratings to be assigned each of the scale factors SF}_{coc(j)}, \text{reducing their collective impact to nil; the 2000 calibration of COCOMO II currently assigns a value of 0.91 to B}_{coc}, \text{which implies that under the best possible system-wide conditions, economies of scale become evident as software increases in size, which is the inverse of what more typically has proven to be the case.} \]
$SF_{coc(i)} =$ the COCOMO II "scale factors" characterizing project conditions that have been shown to have nonlinear impacts on software development effort determining whether economies or diseconomies of scale will likely present during the development.

Note that the definitions of REVL and SCREVL don’t preclude the same lines of absolute custom code reworked due to requirements evolution from also then being reworked due to COTS product volatility. Let’s say you have a program that you delivered with 1000 lines of absolute custom code (not counting any glue code SLOC which recall is not considered “custom” SLOC in COCOTS by definition). But in the process of finishing that program, after creating an initial 1000 lines of custom code, you had to rewrite 100 lines of it due to changes in requirements. Then later you had to rewrite again half of those same 100 lines due to having to remove a COTS component and replace it with a newer version. So to arrive at the final 1000 custom SLOC that you actually delivered, you effectively had to write 1150 lines of custom code (1000+100+50).

Note also that equation 4.8 implies that system level effort due to COTS product volatility exhibits diseconomies of scale as the actual amount of custom code that must be reworked grows. Equation 4.11 implies that system level effort due to COTS product volatility exhibits diseconomies of scale as the relative amount of custom code that must be reworked compared to the amount of effective custom code in the system grows along with actual growth in the effective code.
The total system level effort in the custom code for a project as a result of volatility in the COTS products then becomes a function of the original effort associated with the custom code and a proportional conversion factor AVPF. That factor in turn is varyingly a function of the estimated percentage of rework in the custom code due to COTS volatility SCREVL, the estimated percentage of rework in the custom code due to requirements evolution REVL, the five rated COCOMO II nonlinear scale factors $SF_{COC(i)}$ and the thirteen rated COCOTS glue code submodel linear effort multipliers $EM_{GC}$.

**Total CBS Effort Estimate**

An estimate for the total COTS related effort required in the development of a CBS is then the sum of the effort estimated by the four individual COCOTS submodels:

$$Effort_{COT} = Effort_A + Effort_T + Effort_{GC} + Effort_{SV}$$

*Eq. 4.13*

Finally, an estimate for the grand total effort required to develop a COTS-based software system that incorporates both custom, reuse and off-the-shelf elements thus becomes the combination of the COCOMO II and COCOTS derived portions:

$$Effort_{CBS} = Effort_{COC} + Effort_{COT}$$

*Eq. 4.14*
There were only some preliminary, experimental attempts at calibrating just a few of the parameters in the third pass COCOTS model. And none of them involved the Bayesian techniques described in section 3.2.4. To give a sense of where that incomplete effort stood, the results of one of those preliminary calibration attempts on the glue code submodel using just standard regression techniques on thirteen data points are presented here.

**Glue Code Effort Prediction:**

Based on that partial calibration, for glue code effort prediction 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 after being calibrated on 83 data points using standard regression techniques coupled with a 10% weighted averaging of expert opinion with the empirical data (see section 3.2.4), COCOMO II.1997 came within 30% of actuals 52% of the time. (Recall from tables 2.4 and 3.1, however, that COCOMO II.2000 calibrated with twice as much data and using Bayesian techniques came within 30% of actuals 75% of the time.)
4.1.4 Fourth Iteration

As illustrated in figure 4.5, the fourth pass model represents a moderate restructuring of the COTS estimation model. The biggest single change is the elimination of the volatility submodel.

The system volatility model was problematic. Though the concept of COTS volatility potentially requiring rework effort in the custom coded portions of a CBS is easy to grasp, the proposed equations modeling that effort were admittedly difficult to understand. From a practical point of view, it is also hard to separate out rework effort in the custom code related solely to COTS volatility from rework that
is caused by changes in project requirements. This makes collection of calibration
data for this submodel a difficult task as well.

Fortunately, there was straightforward solution. It turns out that the REVL term
in COCOMO II had already been redefined from its COCOMO 81 counterpart called
"Breakage" to include rework in the custom code as a result of COTS volatility as
well as rework in the custom code due to requirements change (just as does CREVL
in the glue code submodel). This eliminates the need for the system volatility
submodel altogether. This also has the advantage of restoring the accounting for all
custom code effort to COCOMO II, which conceptually seems cleaner. The fourth
iteration version of COCOTS is thus focused strictly on the effort directly associated
with the integration and maintenance of the COTS products alone.

The other significant change from the third pass model was the redefinition of
the glue code scale factor AAREN. It is more reasonable for this parameter to
explicitly adopt the same definition and rating scale as the COCOMO II scale factor
RESL (see section 2.4.1) since both parameters attempt to model essentially the
same concept. It also seems likely that it would be an unusual circumstance for the
two parameters to warrant different ratings during a given CBS development since
they both attempt to characterize the overall architecting process applied to the CBS.
Due to differences in the specific formulas in which they are used, however, they
would both probably still require their own unique numerical parameter values.

The mathematical form for each the remaining remains the same.
The assessment submodel is formulated as follows:

\[ \text{Effort}_A = \text{IFE} + \text{DAE} \quad \text{Eq. 4.15a} \]

where

\[ \text{IFE} = \sum_{i=1}^{m} (\text{CCF}_i \cdot \text{MIFE}_i) \quad \text{Eq. 4.15b} \]

and

\[ \text{DAE} = \sum_{j=1}^{n} (\text{CCD}_j \cdot \text{MDAE}_j) \quad \text{Eq. 4.15c} \]

and

\[ \text{Effort}_A = \text{total COTS product assessment effort for the project.} \]

\[ \text{IFE} = \text{total initial filtering effort.} \]

\[ \text{DAE} = \text{total detailed assessment effort.} \]

\[ M = \text{number of different classes of COTS products going through initial filtering.} \]

\[ N = \text{number of different classes of COTS products going through detailed assessment.} \]

\[ \text{CCF}_i = \text{number of candidate COTS products within a given class of products going through initial filtering.} \]

\[ \text{CCD}_j = \text{number of candidate COTS products within a given class of products going through detailed assessment.} \]

\[ \text{MIFE}_i = \text{mean initial filtering effort for a given class of COTS products.} \]

\[ \text{MDAE}_j = \text{mean detailed assessment effort for a given class of COTS products.} \]

The total COTS product assessment activity on a project again becomes the combination of the two kinds of effort, represented by equation 4.15a.
Tailoring

The tailoring submodel takes on this mathematical form:

\[
\text{Effort}_T = \sum_{i=1}^{m} (NCT_i \cdot MTE_i \cdot TCQ_i) \quad \text{Eq. 4.16}
\]

where

\( \text{Effort}_T = \) total COTS product tailoring effort for the project.

\( M = \) number of different classes of COTS products being tailored.

\( NCT_i = \) number of COTS products within a given class of products being tailored.

\( MTE_i = \) mean tailoring effort for a given class of COTS products.

\( TCQ_i = \) tailoring complexity qualifier; a linear multiplier that either increases or decreases the nominal tailoring effort for the given class of products based upon characterizations of the complexity of the tailoring required (the nominal value is 1.0).

Again the formulation of the tailoring submodel presumes that the expected difficulty of the tailoring work that is going to be required to get a given COTS product integrated into a system can be characterized by standardized rating criteria, represented by the parameter \( TCQ \) in equation 4.16 and which is still rated on the same five-point scale (see appendix B).

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.
The glue code submodel takes on this mathematical form:

\[
\text{Effort}_{GC} = A_{GC} (E_{size_{GC}})^{E_{GC}} \prod_{i=1}^{13} EM_{GC(i)}
\]

where

\[
E_{size_{GC}} = Size_{GC} \cdot (1 + \text{CREVL}/100)
\]

and

\[
E_{GC} = B_{GC} + (0.04 \cdot SF_{GC})
\]

and

\[
\text{Effort}_{GC} = \text{glue code development effort in person-months.}
\]

\[
E_{size_{GC}} = \text{effective size of the glue code under development after adjusting for rework that must be done as a result of changes in requirements or volatility in the COTS products for which the glue code is being written.}
\]

\[
Size_{GC} = \text{absolute size of the glue code under development in source lines of code.}
\]

\[
A_{GC} = \text{a multiplicative conversion constant relating glue code size to development effort, now representing the productivity that typically obtains when project conditions allow all thirteen "effort multiplier" parameters EM}_{GC(i)}\text{in the model to be assigned their baseline "nominal" ratings, thus reducing their collective impact to nil.}
\]

\[
EM_{GC(i)} = \text{"effort multipliers" that either increase or decrease the nominal effort estimate given by the equation based upon characterizations of the environmental conditions that exist while the system is under development (nominal = 1.0).}
\]

\[
\text{CREVL} = \text{estimated percentage of glue code that must be reworked during}
\]
development due to changes or evolution in requirements or release of an updated COTS product but explicitly not as a result of programmer error.

\[ E_{GC} = \] an exponential factor that accounts for nonlinear economies or diseconomies of scale that may accrue as glue code increases in size and which in turn is now a function of a constant \( B_{GC} \) and a single scale factor \( SF_{GC} \)

\[ B_{GC} = \] a constant appearing in the exponential factor that represents the costs or savings that still obtain even when project conditions allow the absolute best possible ratings to be assigned to the scale factor \( SF_{GC} \) reducing its impact to nil; until data is available to more precisely calibrate this parameter, set \( = 1 \) in the third pass glue code submodel to reflect the commonly held wisdom that even under the best possible system-wide conditions, economies of scale will not become evident as glue code increases in size.

\[ SF_{GC} = \] a "scale factor" characterizing project conditions that have been shown to have nonlinear impacts on software development effort determining whether economies or diseconomies of scale will likely present during the development.

The thirteen multiplicative cost drivers \( EM_{GC(i)} \) and one scale factor \( SF_{GC} \) defined for the fourth pass glue code submodel are summarized in table 4.5, again followed by their definitions (all of which remain unchanged except for AAREN).
### Table 4.5 – Fourth Pass Glue Code Submodel Parameters

#### Exponential Scale Factor

| AAREN - Application Architectural Engineering |

#### 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 |

#### Effort Multipliers - 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 |

#### Effort Multipliers - Application/System Drivers

| 10) ACREL - Constraints on Application System/Subsystem Reliability |
| 11) AACPX - Application Interface Complexity |
| 12) APER - Constraints on COTS Technical Performance |
| 13) ASPRT - Application System Portability |

The definitions for these drivers are as follows:

Exponential scale factor $SF_{GC}$:

- **Application Architectural Engineering (AAREN):** the percentage of module interfaces specified in the architecture, subjectively averaged with the percentage of known significant risks mitigated through the system architecting process.
5.4 Chapter 5 Endnotes

1 Boehm et al., *Software Cost Estimation with COCOMO II*, pp. 284-291.


6. Bibliography


7. Appendices
Appendix A

Fourth Iteration COCOTS Model Parameter Values
Component classes for which there are no parameter entries either had little or no assessment data available for use in generating individual values; in these cases use the generic component values instead.
Component classes for which there are no parameter entries either had little or no tailoring data available for use in generating individual values; in these cases use the generic component values instead. The TCQ values are currently the same for each component class but with sufficient data could potentially be individualized for each component class.

<table>
<thead>
<tr>
<th>COTS Component Class</th>
<th>MTE&lt;sup&gt;1&lt;/sup&gt;</th>
<th>TCQ&lt;sup&gt;2&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(Person-months)</td>
<td>VL</td>
</tr>
<tr>
<td>1 Generic Component</td>
<td>4.00</td>
<td>0.69</td>
</tr>
<tr>
<td>2 Back office retail</td>
<td>3.00</td>
<td>0.69</td>
</tr>
<tr>
<td>3 Communication</td>
<td>1.00</td>
<td>0.69</td>
</tr>
<tr>
<td>protocols/packages</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 Compilers</td>
<td>-----</td>
<td>0.69</td>
</tr>
<tr>
<td>5 Configuration</td>
<td>-----</td>
<td>0.69</td>
</tr>
<tr>
<td>mgmt/build tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Data conversion</td>
<td>-----</td>
<td>0.69</td>
</tr>
<tr>
<td>packages</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 Databases</td>
<td>38.29</td>
<td>0.69</td>
</tr>
<tr>
<td>8 Device drivers</td>
<td>3.00</td>
<td>0.69</td>
</tr>
<tr>
<td>9 Disk arrays</td>
<td>4.00</td>
<td>0.69</td>
</tr>
<tr>
<td>10 Emulators</td>
<td>3.00</td>
<td>0.69</td>
</tr>
<tr>
<td>11 Engineering tools</td>
<td>-----</td>
<td>0.69</td>
</tr>
<tr>
<td>(reqmt mgmt, design)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 Graphic</td>
<td>-----</td>
<td>0.69</td>
</tr>
<tr>
<td>information system</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 GUIs/GUI builders</td>
<td>14.00</td>
<td>0.69</td>
</tr>
<tr>
<td>14 Middleware</td>
<td>-----</td>
<td>0.69</td>
</tr>
<tr>
<td>15 Network managers</td>
<td>12.67</td>
<td>0.69</td>
</tr>
<tr>
<td>16 Operating systems</td>
<td>2.00</td>
<td>0.69</td>
</tr>
<tr>
<td>17 Problem mgmt</td>
<td>-----</td>
<td>0.69</td>
</tr>
<tr>
<td>18 Report generators</td>
<td>6.00</td>
<td>0.69</td>
</tr>
<tr>
<td>19 Software process</td>
<td>-----</td>
<td>0.69</td>
</tr>
<tr>
<td>tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 Telecommunication</td>
<td>-----</td>
<td>0.69</td>
</tr>
<tr>
<td>&amp; infrastructure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21 Telemetry Analysis</td>
<td>-----</td>
<td>0.69</td>
</tr>
<tr>
<td>22 Telemetry</td>
<td>-----</td>
<td>0.69</td>
</tr>
<tr>
<td>processing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23 Word processing</td>
<td>-----</td>
<td>0.69</td>
</tr>
<tr>
<td>24 Collaborative</td>
<td>3.00</td>
<td>0.69</td>
</tr>
<tr>
<td>tools</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<sup>1</sup>Mean Tailoring Effort  <sup>2</sup>Tailoring Complexity Qualifier
### Table A.3 – Glue Code Submodel Parameter Values

<table>
<thead>
<tr>
<th></th>
<th>( A_{GC} ) (^1)</th>
<th>2.13 (person-months/KSLOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( SF_{GC} ) (^2)</td>
<td>VL L N H VH</td>
<td></td>
</tr>
<tr>
<td>AAREN</td>
<td>4.00 3.00 2.00 1.00 0.00</td>
<td></td>
</tr>
<tr>
<td>( EM_{GC(i)} ) (^3)</td>
<td>VL L N H VH</td>
<td></td>
</tr>
<tr>
<td>Personnel Drivers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ACIEP</td>
<td>1.34 1.16 1.00 0.86 0.75</td>
</tr>
<tr>
<td>2</td>
<td>ACIPC</td>
<td>1.60 1.27 1.00 0.79 0.62</td>
</tr>
<tr>
<td>3</td>
<td>AXCIP</td>
<td>1.12 1.00 0.89 0.79</td>
</tr>
<tr>
<td>4</td>
<td>APCON</td>
<td>1.58 1.26 1.00 0.80 0.63</td>
</tr>
<tr>
<td>COTS Component Drivers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ACPMT</td>
<td>1.45 1.20 1.00 0.83 0.69</td>
</tr>
<tr>
<td>6</td>
<td>ACSEW</td>
<td>1.07 1.00 0.94 0.88</td>
</tr>
<tr>
<td>7</td>
<td>APCPX</td>
<td>0.82 1.00 1.22 1.48</td>
</tr>
<tr>
<td>8</td>
<td>ACPPS</td>
<td>1.14 1.00 0.88 0.77</td>
</tr>
<tr>
<td>9</td>
<td>ACPTD</td>
<td>1.20 1.09 1.00 0.91 0.84</td>
</tr>
<tr>
<td>Application/System Drivers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ACREL</td>
<td>0.88 1.00 1.14 1.30</td>
</tr>
<tr>
<td>11</td>
<td>AACPX</td>
<td>0.84 1.00 1.19 1.42</td>
</tr>
<tr>
<td>12</td>
<td>ACPER</td>
<td>1.00 1.11 1.22</td>
</tr>
<tr>
<td>13</td>
<td>ASPRT</td>
<td>1.00 1.07 1.14</td>
</tr>
</tbody>
</table>

\(^1\) Size-to-Effort Conversion Constant  
\(^2\) Exponential Scale Factor  
\(^3\) Linear Effort Multiplier
Appendix B

COCOTS Data Collection Survey
UNIVERSITY OF SOUTHERN CALIFORNIA
CENTER FOR SOFTWARE ENGINEERING

COTS Software Integration Cost Modeling

USC COCOTS Model

Project Level COTS Integration Experience Survey

Prepared by
Chris Abts
(with Betsy Clark)

Version 0.7

3 March 2000
# Table of Contents

I. Introduction................................................................. p. 1

II. Definitions/Glossary...................................................... p. 3

III. Identifying Information................................................ p. 9

IV. Systems Data.............................................................. p. 10

V. COTS Assessment Data.................................................. p. 18

VI. COTS Tailoring Data....................................................... p. 22

VII. COTS Glue Code Data.................................................. p. 25

VIII. Glue Code Development Cost Drivers......................... p. 29

IX. Application Effort Due to COTS Volatility Data. p. 46

X. Free Form Comments..................................................... p. 48
I. Introduction

Purpose and Scope

The goal of this survey is to capture from software development organizations specific COTS and NDI product integration experience at the project level; that is, at the level at which the integration work is actually being performed. This information will be used to refine and calibrate the Constructive COTS Integration Cost (COCOTS) model currently under development at the University of Southern California. The intent of this modeling effort is to create a tool which will be able to reasonably predict the expected initial cost of integrating COTS and NDI software into a new software system development or system refresh. COCOTS is a broad-based model, currently focusing on four major sources of integration costs: 1) COTS product assessment, 2) COTS product tailoring, 3) Integration or "glue code" development, and 4) Application volatility due to use of COTS products. It currently does not account for costs associated with licensing, life cycle issues beyond initial development, or post initial development maintenance issues. While significant, these costs will be addressed in later stages of the COCOTS modeling effort.

Disclaimer

NO INFORMATION DIVULGED IN THIS SURVEY WILL BE USED BY ANY AGENCY OF THE FEDERAL GOVERNMENT OR ANY OTHER ORGANIZATION EITHER PUBLIC OR PRIVATE FOR THE PURPOSES OF SOURCE SELECTION, OR EVALUATIVE ASSESSMENT OF RESPONDING UNITS OR ORGANIZATIONS IN ANY FASHION.

Execution

This is a detailed survey. We are looking for as much accuracy as possible, because we intend to use this data to calibrate our model, making consistent and accurate responses crucial. To that end, we would prefer the return of empirical project effort, productivity, and sizing data over best "educated estimates," though we recognize the latter may be all that is available in some circumstances. With that in mind, it is suggested that you quickly read through all the survey questions once before trying to answer them so you can decide beforehand whether you might need some help gathering the information requested in some of the questions.

Please record your answers to the survey directly on this form. We ask that you make the best effort possible to provide an answer to all the questions. If you are
unsure of an answer, or feel a question does not apply to your project, please indicate so rather than leave a question blank.

Completed forms should be returned to the contact identified below under Survey Point of Contact/Data Submission.

Time Required

Based upon pilot executions of the survey, WITH THE DATA AT HAND, it is expected that an informed individual should need on average no more than two hours to complete the questionnaire. However, gathering and preparing the information needed to complete the survey could potentially require SIGNIFICANTLY MORE TIME, perhaps even some part time effort spread over several days or weeks. Please keep this in mind if you have been asked to return this survey within a specific time frame. Again, please skim through the survey quickly one time upon first receiving it to decide how much time you will require to gather the necessary information. Also, note that some sections ask for more detailed data than others. This is a reflection of the weight placed on contribution to overall COTS integration costs. Those areas with greater overall impact are examined in greater detail.

Survey Point of Contact/Data Submission

To return the completed survey, or if you have questions about it, or desire assistance in filling it out, please contact either:

Mr. Chris Abts  
Center for Software Engineering  
University of Southern California  
Henry Salvatori Hall Room 328  
Los Angeles, California 90089-0781  
tel: 213.740.6470  
fax: 213.740.4927  
e-mail: cabts@sunset.usc.edu

Dr. Elizabeth (Betsy) Bailey Clark  
Software Metrics, Inc.  
4345 High Ridge Road  
Haymarket, Virginia 20169  
tel: 703.754.0115  
fax: 703.754.0115  
e-mail: BetsyClark@erols.com

COCOTS Information on the Web:  
http://sunset.usc.edu/COCOTS/cocots.html
II. Definitions/Glossary

**Application Volatility** - creates difficulty in benchmarking stable system configurations, resulting from the use of COTS products which may experience multiple or frequent product releases or upgrades during system development.

**Attribute** - characteristic of a COTS package or associated products and services which are evaluated and used in comparing alternative products and as input into a buy/no buy decision.

**COTS Assessment** - the activity of determining the appropriateness or feasibility of using specific COTS products to fulfill required system functions.

**COTS software** - “commercial-off-the-shelf” software commercially available as stand-alone products and which offer specific functionality needed by a larger system into which they might be incorporated. Generally there is no access to source code for COTS products, which are treated as black boxes with application program interfaces. (In some cases, however, some access to COTS source code is available, in which case these products have been described as “gray” or “white” box COTS.)

**COTS Tailoring** - the activity associated with setting or defining shell parameters or configuration options available for a COTS product, but which do not require modification of COTS source code, including defining I/O report formats, screens, etc.

**NDI software** - “non-developmental item” software available from some source other than the organization developing the system into which the NDI component is to be integrated. The source can be commercial, private, or public sector, just so long as the procuring organization expended no resources on the NDI component’s initial development. Source code is usually available for an NDI component, which may or may not be able to function as a stand-alone item.

**Integration or “glue” code** - software developed in-house and composed of 1) code needed to facilitate data or information exchange between the COTS/NDI component and the system or other COTS/NDI component into which it is being integrated, 2) coded needed to connect or “hook” the COTS/NDI component into the system or other COTS/NDI component but does not necessarily enable data exchange, and 3) code needed to provide required functionality missing in the COTS/NDI component AND which depends upon or must interact with the COTS/NDI component.
AA Percentage of reuse effort due to assessment and assimilation

AAF Adaptation Adjustment Factor

AAM Adaptation Adjustment Multiplier

ASLOC Adapted Source Lines of Code

BRAK Breakage. The amount of controlled change allowed in a software development before requirements are "frozen."

CASE Computer Aided Software Engineering

CM Percentage of code modified during reuse

COCOMO Constructive Cost Model

Cost Driver A particular characteristic of the software development that has the 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

DI Degree of Influence

DM Percentage of design modified during reuse

ESLOC Equivalent Source Lines of Code

FP Function Points

GFS Government Furnished Software

IM Percentage of integration redone during reuse

KASLOC Thousands of Adapted Source Lines of Code

KESLOC Thousands of Equivalent Source Lines of Code

KSLOC Thousands of Source Lines of Code

PM Person Months. A person month is the amount of time one person spends working on the software development project for one month.

SEI Software Engineering Institute
ASSESSMENT ATTRIBUTE DEFINITIONS:

CORRECTNESS

Accuracy - The freedom of system output from error.

Correctness - The degree to which a COTS component is free from faults in its specification, design, and implementation.

AVAILABILITY/ROBUSTNESS

Availability - The degree to which a COTS component is operational and accessible when required for use. Often expressed as a probability.

Fail safe - Pertaining to a COTS component that automatically places itself in a safe operating mode in the event of a failure.

Fail soft - Pertaining to a COTS component that continues to provide partial operational capability in the event of certain failures.

Fault tolerance - Pertaining to a COTS component that is able to continue normal operation despite the presence of faults.

Input Error tolerance - The ability of a COTS component to continue normal operation despite the presence of erroneous inputs.

Redundancy - The presence of auxiliary components in a system to perform the same or similar functions as other elements for the purpose of preventing or recovering from failures.

Reliability - The ability of a COTS component to perform its required functions under stated conditions for a specified period of time; the probability that a COTS component will perform its intended functions satisfactorily for a prescribed time and under stipulated conditions.

Robustness - The degree to which a COTS component can function correctly in the presence of invalid inputs or stressful environmental conditions.

Safety - Protection against software or hardware faults that could result in harm to people, data or systems.
SECURITY

Security (Access Related) - the degree to which a system or component prevents unauthorized access to, or modification of, computer programs or data.

Security (Sabotage Related) - Protection against exploitable weaknesses that could result in harm to people, data, or systems.

PRODUCT PERFORMANCE

Execution Performance - The degree to which a COTS component performs its functions within given execution timing constraints.

Information/data Capacity - The quantity of information or logical data items that can be stored or maintained by a system or COTS component relative to the expected needs of the users.

Precision - The degree of exactness or discrimination with which a quantity is stated; for example, a precision of 2 decimal places versus a precision of 5 decimal places.

Memory Performance - The degree to which a COTS component performs its functions within given memory constraints (hard storage and/or virtual storage).

Response time - The elapsed time between the end of an inquiry or command to an interactive computer system and the beginning of the system’s response.

Throughput - The amount of work that can be performed by a COTS component in a given period of time, for example, number of jobs per day.

UNDERSTANDABILITY

Documentation quality - The degree to which a COTS component contains enough information to explain its objectives, operations, properties and other attributes to be useful in understanding, tailoring, verifying, and operating it.

Simplicity - The degree to which a COTS component has a design and implementation that is straightforward and easy to understand.

Testability - The degree to which a COTS component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.

EASE OF USE

Usability/Human Factors - The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.
VERSION COMPATIBILITY

Downward compatibility - Pertaining to software that is compatible with an earlier or less complex
version of itself, for example, a COTS component that handles files created by an earlier version of
itself.

Upward compatibility - Pertaining to software that is compatible with a later or more complex
version of itself, for example, a COTS component that handles files created by a later version of itself.

INTERCOMPONENT COMPATIBILITY

Compatibility with other components - The ability of two or more components to perform their
required functions while sharing the same hardware or software environment.

Interoperability - The ability of two or more systems or components to exchange information and to
use the information that has been exchanged.

FLEXIBILITY

Extendability - The ease with which features can be added to or around a COTS component in order
to increase storage or functional capability.

Flexibility - The ease with which a COTS component can be tailored for use in applications or
environments other than those for which it was specifically designed or is normally used.

INSTALLATION/UPGRADE EASE

Installation ease - The ease with which a COTS component can be installed within a hardware or
software environment.

Upgrade/refresh ease - The ease with which a new version of a COTS component can be installed
within a hardware or software environment.

PORTABILITY

Portability - The ease with which a COTS component can be transferred from one hardware or
software environment to another.

FUNCTIONALITY

Functionality - The degree to which a COTS component has the functional capability needed by a
user to solve a problem or achieve an objective; a functional capability that must be met or possessed
by a COTS component to satisfy a set of requirements.

PRICE

Initial purchase or lease – The upfront cost to buy or lease a COTS component.
Recurring costs – The periodic (usually annual) cost for maintenance and other COTS-related support.

MATURETY

Product Maturity - The length of time that a COTS component has been commercially available and/or the size and diversity of its user base.

Vendor Maturity - The length of time that a vendor has been in the COTS software business and/or the size and diversity of its user base.

VENDOR-SUPPORT

Response time for critical problems - The speed with which critical problems are addressed and solutions are put in place by the vendor.

Support - Responsiveness in answering user questions, and in dealing with user problems in installing, testing, and using the COTS component.

Warranty - The vendor’s written guarantee that the product will perform as specified and that instances of non-compliance will be resolved according to a written agreement between the vendor and the buyer.

TRAINING

User training - The degree to which vendor training results in users who are proficient in using the COTS component to solve problems or accomplish objectives.

VENDOR CONCESSIONS

Willingness to escrow source code - Willingness of the vendor to place the source code in the hands of a third-party, thereby providing protection to the procurer in the event that the vendor goes out of business or stops supporting the COTS component.

Willingness to Make Modifications - Willingness to make and maintain procurer-specific modifications to the COTS product, rather than being driven solely by general market demands.
III. Identifying Information

Date Survey Completed: ________________________________

Organization Name: ___________________________________

Name of Preparer: _____________________________________

Title of Preparer: _____________________________________

Voice Phone Number: ___________________________________

Fax Number: ___________________________________________

E-mail Address: _________________________________________

Postal Address: _________________________________________

_____________________________________________________

_____________________________________________________

_____________________________________________________

_____________________________________________________
IV. Systems Data

4.1 Project Domain

Circle one:

**Core System Functionality**
- Operational, Mission Critical
- Operational, Non-mission Critical
- Support (e.g., software development tools, logistical planning, etc.)

**Communications, Navigation, and Surveillance**
- Operational, Mission Critical
- Operational, Non-mission Critical
- Support (e.g., software development tools, logistical planning, etc.)

**Administrative**
- Operational, Business Critical
- Operational, Non-business Critical
- Support (e.g., software development tools, logistical planning, etc.)

**Other**
- Describe:
4.2 Development Type

Circle one: New System System Upgrade/Refresh

4.3 Development Process

Circle one: Waterfall Spiral

Other: ______________________

(Please describe on back of this page.)

4.4 Iteration

If the development process reported in 4.3 is iterative (e.g., spiral), indicate the iteration being reported.

Iteration: ____________________
4.5 Current Project Phase or Activity

Report the development phase or activity the project is currently undergoing based upon one of the following three schemes:

1) It is recognized that there is always some overlap between activities even if a waterfall process is being followed but you are being asked here for the phase or activity which is the current major focus of the project. Circle that one phase.

2) It is also recognized that some modern development processes—particularly those involving COTS/NDI components—perform certain activities concurrently. For example, Requirements Definition and COTS/NDI Evaluation & Assessment may be undertaken together because system requirements will influence the suitability of given COTS/NDI components, but also the COTS/NDI components that are currently available or on the market may help determine final system requirements. If this is the case, circle the phrase “concurrent phasing” and circle all the major activities the project is currently undergoing simultaneously.

3) Finally, if you report that the project is currently not undergoing any development phase or activity because it is completed or in maintenance, it is assumed that the development phases for this completed project include Software Requirements through Integration/Test, including COTS/NDI Assessment. If this is not the case, please describe the correct phasing on the back of this page.

Circle one or more as needed:

Concurrent Phasing

Requirements Definition
COTS/NDI Assessment
Design
System Coding/COTS Integration
Unit Test
System Integration/Test
Development Completed
Maintenance

Where in the life-cycle does COTS/NDI assessment occur (For example, prior or post requirements definition)?:__________________
4.6 Delivery Scheduling

Report the nature of the project delivery scheduling.

Circle one:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>Delivery acceptance required at one location, no on-going maintenance</td>
<td>Delivery acceptance required at one location, maintenance on-going</td>
<td>Phased delivery acceptance required at more than one location, no on-going maintenance</td>
<td>Phased delivery acceptance required at more than one location, maintenance on-going</td>
</tr>
</tbody>
</table>

4.7 Project Time Frame

1) For completed projects record the year of completion (i.e. the year of final delivery acceptance); or 2) for projects in maintenance record the year of initial delivery acceptance; or 3) for projects with a phased delivery schedule, the year of first site delivery acceptance; or 4) for projects still in development indicate the current year.

Year:__________ Year determined by criteria (circle one): 1 2 3 4

4.8 Schedule Duration

Record the number of calendar months from the time the development began (i.e. the start of system requirements definition) through either 1) the date of final delivery acceptance if the project was completed, 2) delivery to the first site if the project was/is undergoing a phased delivery of copies at multiple locations, 3) the date of initial delivery acceptance if the project is in maintenance, or 4) the current date if the project is still in development.

Months:__________ Months determined by criteria (circle one): 1 2 3 4
4.9 Development Schedule Life-cycle Phases

Circle all the life-cycle phases the schedule reported in 4.8 covers:

<table>
<thead>
<tr>
<th>System Requirements</th>
<th>COTS/NDI Assessment</th>
<th>Detailed Design</th>
<th>System Integration and Test</th>
<th>Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Requirements</td>
<td>Preliminary Design</td>
<td>Code/ COTS Integration and Unit Test</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.10 Project Total Effort

Record the total effort expended in Person-Months from the time the development began (i.e. the start of system requirements definition) through either 1) the date of final delivery acceptance if the project was completed, 2) the date of initial delivery acceptance, and then from initial delivery acceptance to the current date if the project is in maintenance, 3) the date of delivery acceptance to the first site if multiple copies of the system have a phased installation schedule to multiple locations, or 4) the current date if the project is still in development.

Development Person-Months:__________  (Maintenance Phase Person-Months:__________)

Person-months determined by criteria (circle one): 1 2 3 4

4.11 Development Effort Life-cycle Phases

Circle all the life-cycle phases the total effort reported in 4.10 covers:

<table>
<thead>
<tr>
<th>System Requirements</th>
<th>COTS/NDI Assessment</th>
<th>Detailed Design</th>
<th>System Integration and Test</th>
<th>Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Requirements</td>
<td>Preliminary Design</td>
<td>Code/ COTS Integration and Unit Test</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Note: items 4.9 and 4.11 may seem redundant, but sometimes schedule and effort data are not available over precisely identical life-cycle phases.)
4.12 Standard Person-Month

Record the average number of work hours defining a person-month in your organization.

Hours per Person-Month: ________________

4.13 Total Delivered Source Code

Record the total number of lines of source delivered at project completion (or generated to date if the project is still in development or in maintenance), including NDI code, and including COTS glue code. (This question is intended to provide us with a sense of scale of your overall system development.)

Total SLOC: ________________

4.14 SLOC Count Type

Record the unit definition for the SLOC count reported in 4.13.

Circle one:

- Logical SLOC
- Physical SLOC (carriage returns)
- Physical SLOC (semicolons)
- Non-commented/Non-blank SLOC
- Other: ________________________________
4.15 Programming Languages

Record all the various programming languages used in the development and the percentage of the SLOC reported in 4.13 representing each language.

<table>
<thead>
<tr>
<th>Language</th>
<th>Percent SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.16 **Total System Function Points**

Record the total number of unadjusted function points counted for system at project completion (or to date if still in development or in maintenance), including NDI functionality, and including COTS glue functionality. (Again, This question is intended to provide us with a sense of scale of your overall system development, this time using an alternate measure.)

Total FP: ______________

4.17 **System Architecture**

Record the nature of the overall system architecture. If the architecture is essentially uniform or homogenous, circle the one style descriptor below which best describes that architecture. If the architecture is substantially a mix of multiple architectural styles, circle as many style descriptors as needed to describe the overall system architecture.

Circle as needed:

<table>
<thead>
<tr>
<th>Pipe &amp; Filter</th>
<th>Distributed</th>
<th>Main/Subroutine</th>
<th>Event Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multithreaded</td>
<td>Blackboard/</td>
<td>Closed Loop</td>
<td>Real Time</td>
</tr>
<tr>
<td></td>
<td>Single Layer</td>
<td>Feedback Control</td>
<td></td>
</tr>
<tr>
<td></td>
<td>General Repository</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rule Based</td>
<td>Transactional</td>
<td>Layered</td>
<td>Other: ______</td>
</tr>
<tr>
<td></td>
<td>Database Centric</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.18 **System Architecting Process**

Describe the process which was followed to arrive at the system architecture recorded in 4.17 in the space below. (For example, was there a paper analysis relative to project specifications, at least on the highest risk system elements? Was any prototyping performed or simulations on performance issues conducted? Were formal Architectural Review Boards used? Or was no formal architecting process used at all?):
V. COTS Assessment Data

The COTS Assessment sub-model presumes COTS component assessment is done in a two-pass manner. The first pass is a “quick and dirty” activity in which minimal effort is expended, based mainly on engineering judgment of vendor supplied product specifications, and designed to rapidly remove from consideration those COTS products which on the face are not viable candidates for further consideration. The second pass is a more careful examination of the remaining COTS candidates, evaluating each product according to certain desirable attributes.

In this section, the data being requested is **an aggregate** of the total amount of effort expended doing COTS/NDI assessment during the system development.

**Initial Filtering Effort:**

5.1 Total number of candidate COTS products filtered: _________

5.2 Total effort spent doing initial filtering of all COTS candidates: _________(person-months)

5.3 Average filtering effort per COTS candidate: _________(person-hours or person-months)  *circle the correct units*

**Attribute Assessment Effort:**

5.4 Total number of COTS products assessed: _________

5.5 Total number of unique COTS products finally integrated: _________

5.6 Total effort spent on attribute assessment of all COTS candidates: _________(person-months)

5.7 Total number of calendar months spent assessing all candidate COTS products: _________(months)

5.8 For each attribute listed in the table following, indicate the total amount of effort expended assessing COTS products during system development in aggregate in terms of the given attribute by checking the appropriate box according to the column definitions indicated below:

| U | “Unknown” | “don’t know effort expended assessing this attribute.” |
| EL | “Extra Low” | “no effort expended.” |
| VL | “Very Low” | “less than or equal to one person-hour.” |
L - "Low" - more than one person-hour and less than or equal to one person-day.
N - "Nominal" - more than one person-day and less than or equal to one person-week.
H - "High" - more than one person-week and less than or equal to one person-month.
VH - "Very High" - more than one person-month and less than or equal to three person-months.
EH - "Extra High" - more than three person-months and less than or equal to N person-years.

(Please indicate your value of N in person-years: N = )

Note: In the table on the following pages, the composite attributes we want you to rate are highlighted in the shaded rows. Underneath each composite attribute is at least one and sometimes several lower level attributes which are being aggregated into the composite attribute.

Place a check mark in the appropriate box of a given composite attribute to indicate how much effort was expended assessing any or all of the lower level attributes listed under that particular composite attribute.

Also, place a check mark in the first box to the right of the lower level attributes you actually considered when determining the effort expended assessing a given composite attribute.

For example, for "Correctness," you may indicate that the assessment effort was Very High, and you considered both of the lower level attributes accuracy and correctness as defined in the glossary. Thus you would put a check in the VH box in the shaded row for "Correctness," and put checks in both of the boxes immediately to the right of the two lower level attributes.

But moving on to "Availability/Robustness," you may determine only a Nominal amount of effort was expended assessing this composite attribute, and in this case the only lower level attributes considered were availability, fail safe, reliability, and safety. So you would put check marks in the N box in the shaded row for "Availability/Robustness," and check marks in the first box to the right of availability, fail safe, reliability, and safety.

Finally, space has been left at the end of the table for you to specify your own assessment attributes if you find that you have assessment effort that is not accounted for by the existing set of pre-defined attributes.
<table>
<thead>
<tr>
<th>Attributes/Effect</th>
<th>U</th>
<th>E</th>
<th>V</th>
<th>L</th>
<th>N</th>
<th>H</th>
<th>V</th>
<th>H</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Correctness:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correctness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Availability/Robustness:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fail Safe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fail Soft</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Input Error Tolerance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Redundancy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reliability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robustness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safety</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Security:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security (Access Related)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security (Sabotage Related)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Product Performance:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution Performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information/Data Capacity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precision</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory Performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Response Time</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Throughput</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Understandability:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation Quality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simplicity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Ease of Use:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usability/Human Factors</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Version Compatibility:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Downward Compatibility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Upward Compatibility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Intercomponent Compatibility:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compatibility with Other Components</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interoperability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Flexibility:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extendibility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flexibility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Installation/Upgrade Ease:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Installation Ease</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Upgrade/Refresh Ease</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Table continued on following page.)
<table>
<thead>
<tr>
<th>Attributes</th>
<th>Effort</th>
<th>U</th>
<th>E</th>
<th>V</th>
<th>L</th>
<th>N</th>
<th>H</th>
<th>V</th>
<th>H</th>
<th>E</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portability:</td>
<td>Portability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functionality:</td>
<td>Functionality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Price:</td>
<td>Initial Purchase or Lease</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Recurring Costs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Maturity:</td>
<td>Product Maturity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Vendor Maturity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vendor Support:</td>
<td>Response Time for Critical Problems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Support</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Warranty</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training:</td>
<td>User Training</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vendor Concessions:</td>
<td>Willingness to Escrow Source Code</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Willingness to Make Modifications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other (Please define)</td>
<td>1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
VI. COTS Tailoring Data

COTS tailoring involves those normal activities required to prepare or initialize a COTS component for use in a specific system WITHOUT adapting or modifying the normal or available functionality of the COTS component.

(Adapting or modifying functionality, e.g., to account for mismatches between your business process and a COTS product’s functionality, is typically done in the glue code and is handled by the COCOTS glue code sub-model through proper sizing of the glue code effort and rating of the glue code parameters. Large scale database population or conversion is also not considered “tailoring.” This effort is captured in the COCOMO II model with the DATA parameter. However, specification of data definition templates and formats is considered tailoring if this is part of the normal activity needed to initialize a given COTS component.)

Major tailoring activities include parameter specification, script writing, I/O report and GUI screen specification, and set-up of security/user access protocols.

The COTS Tailoring sub-model presumes that aggregate project COTS tailoring activity can be characterized by an overall level of complexity, which in turn has implications for the overall effort expended on tailoring all the COTS components in a system.

Tailoring Effort:

6.1. Total number of COTS components in system tailored: __________

6.2. Total effort spent tailoring all COTS components in system: __________ (person-months)

6.3. Total number of calendar months spent tailoring all COTS components: __________ (months)

Tailoring Activity Complexity:

Complexity of aggregate COTS tailoring activities is determined in the model using a subjective average of the individual complexity of five equally weighted factors (four tailoring activities plus one tailoring aid) presented in table VI.A on the following page. To determine aggregate tailoring complexity, first rate the five factors in table VI.A individually according to the criteria given in the table. (Again,
keep in mind that you are doing a mental averaging of each factor as it was performed or applied across all COTS components in the system.) Next, sum the total point score as described in the table for the combination of ratings you selected. Then determine which gross category that score corresponds to on the rating scale provided on the page following the table. Finally, using your best engineering judgment, adjust your final rating for aggregate complexity above or below the center mark of the gross category as needed.
<table>
<thead>
<tr>
<th>Tailoring Activities &amp; Aids</th>
<th>Very Low (point value = 1)</th>
<th>Low (point value = 2)</th>
<th>Nominal (point value = 3)</th>
<th>High (point value = 4)</th>
<th>Very High (point value = 5)</th>
<th>Corresponding Points</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameter Specification</strong></td>
<td>Zero to 50 parms to be initialized.</td>
<td>51 to 100 parms to be initialized.</td>
<td>101 to 500 parms to be initialized.</td>
<td>501 to 1000 parms to be initialized.</td>
<td>1001 or more parms to be initialized.</td>
<td></td>
</tr>
<tr>
<td><strong>Script Writing</strong></td>
<td>Menu driven; 1 to 5 line scripts; 1 to 5 scripts needed.</td>
<td>Menu driven; 6 to 10 line scripts; 6 to 15 scripts needed.</td>
<td>Hand written; 11 to 25 line scripts; 16 to 30 scripts needed.</td>
<td>Hand written; 26 to 50 line scripts; 31 to 50 scripts needed.</td>
<td>Hand written; 51 or more line scripts; 51 or more scripts needed.</td>
<td></td>
</tr>
<tr>
<td><strong>I/O Report &amp; GUI Screen Specification &amp; Layout</strong></td>
<td>Automated or standard templates used; 1 to 5 reports/screens needed.</td>
<td>Automated or standard templates used; 6 to 15 reports/screens needed.</td>
<td>Automated or standard templates used; 6 to 25 reports/screens needed.</td>
<td>Hand written or custom designed; 26 to 50 reports/screens needed.</td>
<td>Hand written or custom designed; 51 or more reports/screens needed.</td>
<td></td>
</tr>
<tr>
<td><strong>Security/Access Protocol Initialization &amp; Set-up</strong></td>
<td>1 security level; 1 to 20 user profiles; 1 input screen/user.</td>
<td>2 security levels 21 to 50 user profiles; 2 input screens/user.</td>
<td>3 security levels 51 to 75 user profiles; 3 input screens/user.</td>
<td>4 security levels 76 to 100 user profiles; 4 input screens/user.</td>
<td>5 or more security levels 101 or more user profiles; 5 or more input screens/user.</td>
<td></td>
</tr>
<tr>
<td><strong>Availability of COTS Tailoring Tools</strong></td>
<td>Tools were highly useful.</td>
<td>Tools were very useful.</td>
<td>Tools were moderately useful.</td>
<td>Tools were somewhat useful.</td>
<td>No tools available.</td>
<td></td>
</tr>
</tbody>
</table>

**Total Point Score =**

**Table VI.A - Aggregate Tailoring Activity Complexity**
Example: individual ratings of Low for Parameter Spec, Very Low for Scripts, Very High for I/O Reports, Nominal for Security, and Very High for Tailoring Tools would result in a point total of 16, indicating a gross combined rating of Nominal. To recognize the existence of at least one individual rating of Very Low, however, it might be reasonable to circle the tic mark exactly halfway between the Low and Nominal categories on the scale below when assigning a final complexity rating for aggregate COTS tailoring activity. Note that the minimum point total possible is 5 and the maximum is 30.

6.4 - Aggregate Tailoring Complexity.

Circle the appropriate tic mark based upon the criteria in the preceding table:

<table>
<thead>
<tr>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point total is between 5 and 10.</td>
<td>Point total is between 11 and 15.</td>
<td>Point total is between 16 and 20.</td>
<td>Point total is between 21 and 25.</td>
<td>Point total is between 26 and 30.</td>
</tr>
</tbody>
</table>

Explain rationale for your rating:
VII. COTS Glue Code Data

The COTS Glue Code model refers to the total amount of COTS glue code developed for the system in aggregate, including glue code created for COTS products layered beneath other COTS products, as opposed to being integrated directly into the main system.

7.1 Total number of COTS components represented by glue code described in this section.

No. Components: __________

7.2 All functions provided by the COTS components counted in question 7.1.

Circle as needed:

- Spreadsheet
- Word Processing
- Scheduling
- Mathematical Utilities
- Other:

- Communications
- User Display
- Database
- Signal Processing
- Message Handling
- CASE Environment
- Diagnostics
- Compiler

7.3 Component Integration Nature

Indicate the percentage of COTS components counted in question 7.1 for which the integration activity is a:

New Component Integration: ________%

Component Upgrade/Refresh: ________%
7.4 Specific Glue Code Development Activity Duration

Record the number of calendar months needed to complete all the glue code from the time the first COTS component counted in question 7.1 was integrated to the last COTS component.

Months: ____________

7.5 Specific Glue Code Development Activity Effort

Record the total effort expended in Person-Months needed to complete all the glue code from the time the first COTS component counted in question 7.1 was integrated to the last COTS component.

Person-Months: ____________

We would like to collect integration or "glue" code sizing and breakage data in physical and logical lines of code as well as unadjusted function points. Please submit all size measures that are available.

Note: Glue code as defined for the purposes of this survey is composed of 1) code needed to facilitate data or information exchange between the COTS/NDI component and the system into which it is being integrated, 2) coded needed to connect or "hook" the COTS/NDI component into the system but does not necessarily enable data exchange, and 3) code needed to provide required functionality missing in the COTS/NDI component AND which depends upon or must interact with the COTS/NDI component.
7.6 Total Delivered Lines of Component Glue Code

Record the total number of lines of glue code delivered for all COTS components.

Glue SLOC:__________

7.7 Glue SLOC Count Type

Record the unit definition for the SLOC count reported in question 7.6.

Circle one:

<table>
<thead>
<tr>
<th>Logical SLOC</th>
<th>Physical SLOC (carriage returns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical SLOC (semicolons)</td>
<td>Non-commented/Non-blank SLOC</td>
</tr>
<tr>
<td>Other:</td>
<td>________________________________</td>
</tr>
</tbody>
</table>

7.8 Glue Code Programming Languages

Record all the various programming languages used in the development of the glue code and the percentage of the SLOC reported in question 7.6 representing each language.

<table>
<thead>
<tr>
<th>Language</th>
<th>Percent SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7.9 Glue Code by Unadjusted Function Points

Record the total number of unadjusted function points attributable to data function types (internal logical files, external interface files) and transaction function types (external inputs, external outputs, external inquiries) counted for the glue code delivered with all COTS components.

Glue UFP: ______________

7.18 Glue Code Breakage

Record the percentage breakage in the glue code that occurred by project completion. “Breakage” is defined to be code that had to be discarded or reworked as a result of a change in system requirements OR the need to integrate a newer or upgraded version of a COTS product. It does NOT include code that had to be reworked as result of bugs found during testing or improper implementation of requirements.

Percentage Breakage (Glue SLOC): ________

Percentage Breakage (Glue UFP): ________
VIII. Glue Code Development Cost Drivers

These drivers should be assessed while considering the total amount of COTS glue code developed for the system, as described in Section VII. That is, where the criteria given below refer to “a COTS component,” think in terms of how a given driver would most accurately be rated when considering all COTS components taken together. You’re doing a kind of mental averaging here across all COTS components integrated. The key here is to remember that you are trying to qualify the average or overall conditions that obtained when all the glue code was being developed, whether or not that glue code was written to accommodate only one or many different COTS components.

Fourteen Cost Drivers have been defined for the COTS integration cost estimation model. You are asked to rate each driver according to a specific metric defined for each driver on a scale ranging from Very Low to Very High as the given metric applies to the circumstances of the component integration effort being reported. Descriptions of the concepts being captured by each driver have been provided to help you make your assessment. (Note that these descriptions are usually more encompassing than the specific metric by which you are asked to make your rating.) Also, a graduated scale has been provided to allow you to make incremental ratings between the five gross ratings. Record your answers by circling the tic marks on the scales, one mark per cost driver. Note that some of the cost drivers do not allow ratings at all levels (i.e., Very Low, Low, etc.). Finally, for each question in this section, the word “UNKNOWN” has been placed just above each rating scale. If for any driver you do not know with any reasonable confidence the appropriate rating and cannot determine that information, please circle “UNKNOWN” rather than indicating either nominal or some other rating on the scale.
Circle the appropriate tic mark on the scales below:

**Integration Personnel Drivers**

8.1 ACIEP - COTS/NDI Integrator Experience with Product

How much experience did/does the development staff have with running, integrating, and maintaining the COTS/NDI products?

*Metric: months/years of experience with product.*

**UNKNOWN**

<table>
<thead>
<tr>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Staff on average has no experience with the products.</td>
<td>Staff on average has less than 6 month’s experience with the products.</td>
<td>Staff on average has between 6 month’s and 1 year’s experience with the products.</td>
<td>Staff on average has between 1 and 2 years’ experience with the products.</td>
<td>Staff on average has more than 2 years’ experience with the products.</td>
</tr>
</tbody>
</table>

VL L N H VH

Explain rationale for your rating:
8.2 ACIPC - COTS/NDI Integrator Personnel Capability

What were/are the overall software development skills and abilities which your team as a whole on average brought/bring to the product integration tasks AS WELL AS experience with the specific tools, languages, platforms, and operating systems used/being used in the integration tasks?

*Metric: months/years of experience.*

**UNKNOWN**

<table>
<thead>
<tr>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Staff on average has no development experience or with the specific environmental items listed.</td>
<td>Staff on average has less than 6 month's development experience or with the specific environmental items listed.</td>
<td>Staff on average has between 6 month's and 1 year's development experience or with the specific environmental items listed.</td>
<td>Staff on average has between 1 and 2 years' development experience or with the specific environmental items listed.</td>
<td>Staff on average has more than 2 years' development experience or with the specific environmental items listed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VL</th>
<th>L</th>
<th>N</th>
<th>H</th>
<th>VH</th>
</tr>
</thead>
</table>

Explain rationale for your rating:
8.3 AXCIP - Integrator Experience with COTS/NDI Integration Processes

Does a formal and validated COTS/NDI integration process exist within your organization and how experienced was/is the development staff in that formal process?

*Metric: a mix of conditions including SEI CMM level, ISO 9001 certification, and number of times integration team as a whole on average has used the defined COTS/NDI integration process.*

**UNKNOWN**

<table>
<thead>
<tr>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMM level =1</td>
<td>[CMM level =2</td>
<td>[CMM level =3</td>
<td>[CMM level &gt; 3</td>
</tr>
<tr>
<td>OR</td>
<td>OR ISO 9001 certified]</td>
<td>OR ISO 9001 certified]</td>
<td>OR</td>
</tr>
<tr>
<td>there is no formally</td>
<td>AND</td>
<td>AND</td>
<td>OR</td>
</tr>
<tr>
<td>defined COTS/NDI</td>
<td></td>
<td>there is a formally defined COTS/NDI</td>
<td>ISO 9001 certified]</td>
</tr>
<tr>
<td>integration process</td>
<td></td>
<td>integration process</td>
<td>AND</td>
</tr>
<tr>
<td>AND</td>
<td></td>
<td>AND</td>
<td>there is a formally defined COTS/NDI</td>
</tr>
<tr>
<td>the integration team</td>
<td></td>
<td></td>
<td>integration process</td>
</tr>
<tr>
<td>has never used the</td>
<td></td>
<td></td>
<td>AND</td>
</tr>
<tr>
<td>process before.</td>
<td></td>
<td></td>
<td>the integration team</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>has used the process</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 or 2 times before.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**L** **N** **H** **VH**

Explain rationale for your rating:
8.4 APCON - Integrator Personnel Continuity

How stable was/is your integration team? Are the same people staying around for the duration of the tasks, or must you keep bringing in new people and familiarizing them with the particulars of the project because experienced personnel leave?

*Metric: annual integration personnel turnover rate (a high personnel turnover rate implies a low personnel continuity).*

**UNKNOWN**

<table>
<thead>
<tr>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>48% or more per year</td>
<td>Between 24% and 47% per year</td>
<td>Between 12% and 23% per year</td>
<td>Between 6% and 11% per year</td>
<td>5% or less per year</td>
</tr>
</tbody>
</table>

**VL** | **L** | **N** | **H** | **VH**

Explain rationale for your rating:
COTS/NDI Component Drivers

8.5 ACPMT - COTS/NDI Product Maturity

How many copies have been sold or used previously of the major versions (as opposed to release of those versions) of the COTS/NDI components you integrated or intend to integrate? How long have the versions been on the market or available for use? How large are the versions’ market shares or installed user bases? How thoroughly have the versions been used by others in the manner you used or intend to use them?

Metric: time on market (if COTS)/time available for use (if NDI).

<table>
<thead>
<tr>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versions in pre-release beta test.</td>
<td>Versions on market/available less than 6 months.</td>
<td>Versions on market/available between 6 months and 1 year.</td>
<td>Versions on market/available between 1 and 2 years.</td>
<td>Versions on market/available more than 2 years.</td>
</tr>
</tbody>
</table>

VL L N H VH

Explain rationale for your rating:
8.6 ACSEW - COTS/NDI Supplier Product Extension Willingness

How willing were/are the suppliers of the COTS/NDI products to modify the design of their software to meet your specific needs, either by adding or removing functionality or by changing the way it operates? In the case of COTS components, this refers to changes that would appear in market releases of the product. In the case of NDI components, this refers to changes that would appear in copies being distributed to all users of the component. This does NOT include specialty changes in the COTS/NDI component that would appear in your copy only.

*Metric: number and nature of changes supplier will make.*

<table>
<thead>
<tr>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suppliers will not make any changes.</td>
<td>Suppliers will make a few minor changes.</td>
<td>Suppliers will make one major change and several minor ones.</td>
<td>Suppliers will make two or more major changes along with any minor changes needed.</td>
</tr>
</tbody>
</table>

L N H VH

Explain rationale for your rating:
8.7 APCPX - COTS/NDI Product Interface Complexity

What are the nature of the interfaces between the COTS/NDI components and the glue code connecting them to the main application? Are there difficult synchronization issues? Must the interfaces balance conflicting criteria (e.g., security, safety, accuracy, ease of use, speed)?

**Metric:** the scale for this driver uses a subjective average of the three equally weighted facets of interface complexity described in table VII.A on the following page. To rate this driver, first rate the three items (interface conventions, control aspects, data) in table VIII.A individually according to the criteria given in the table. Next, sum the total point score as described in the table for the combination of ratings you selected, and determine which gross category that score corresponds to on the scale below. Finally, using your best engineering judgment, adjust your final rating for this driver above or below the center mark of the gross category as needed.

Example: individual ratings of Low for Interface, Low for Control, and Very High for Data would result in a point total of 9, indicating a gross combined rating of Nominal. To recognize the existence of at least one individual rating of Very High, however, it might be reasonable to circle the tic mark immediately to the right of the center tic mark in the Nominal category on the scale below when assigning a final rating for this driver.

### UNKNOWN

<table>
<thead>
<tr>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point total is between 5 and 7.</td>
<td>Point total is between 8 and 10.</td>
<td>Point total is between 11 and 13.</td>
<td>Point total is between 14 and 15.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>N</th>
<th>H</th>
<th>VH</th>
</tr>
</thead>
</table>

Explain rationale for your rating:
## Table VIII.A - Facets of Complexity

Use this table in evaluating complexity drivers 8.7 (APCPX) and 8.11 (AACPX). Use it once for APCPX, then repeat its use for AACPX. Rate each complexity element described in the table individually, recording the point value associated with your rating in the far right column. Then sum all three point values to arrive at a total point score (minimum score possible is 5, maximum score is 15). Then apply that total point score to the scales provided for each of the two cost drivers as indicated under their descriptions.
8.8 ACPPS - COTS/NDI Supplier Product Support

What is the nature of the technical support for the COTS/NDI components that was/is available AND PROCURED for the integration team during the development, either directly from the component suppliers or through third parties?

*Metric: the level of support available and procured.*

**UNKNOWN**

<table>
<thead>
<tr>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Products are unsupported.</td>
<td>Help desk support</td>
<td>Trained technical</td>
<td>Formal consulting help.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>support.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>N</th>
<th>H</th>
<th>VH</th>
</tr>
</thead>
</table>

Explain rationale for your rating:
8.9 **ACPTD - COTS/NDI Supplier Provided Training and Documentation**

How much training and/or documentation for the COTS/NDI components was/is available AND PROCURED for the integration team during the development, either directly from the component suppliers or through third parties?

*Metric: the amount of training and/or documentation available and procured.*

<table>
<thead>
<tr>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>No training and very little documentation procured.</td>
<td>Roughly ¼ of the needed training and/or documentation procured.</td>
<td>Roughly ½ of the needed training and documentation procured.</td>
<td>Roughly ¾ of the needed training and/or documentation procured.</td>
<td>As much training and/or documentation procured as needed.</td>
</tr>
</tbody>
</table>

### Rating Scale

- **VL** (Very Low)
- **L** (Low)
- **N** (Nominal)
- **H** (High)
- **VH** (Very High)

Explain rationale for your rating:
APPLICATION/SYSTEM Drivers

8.10 ACREL - Constraints on System/Subsystem Reliability

How severe are the overall reliability constraints on the system or subsystem into which the COTS/NDI components was/is being integrated? What are the potential consequences if the components fail to perform as required in any given time frame? (Note that availability is considered an issue different than reliability and is NOT addressed in this cost driver.)

*Metric: the potential threat if the component fails to perform as expected.*

**UNKNOWN**

<table>
<thead>
<tr>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Threat is low; if a failure occurs losses are easily recoverable (e.g., document publishing).</td>
<td>Threat is moderate; if a failure occurs losses are fairly easily recoverable (e.g., support systems).</td>
<td>Threat is high; if a failure occurs the risk is to mission critical requirements.</td>
<td>Threat is very high; if a failure occurs the risk is to safety critical requirements.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>N</th>
<th>H</th>
<th>VH</th>
</tr>
</thead>
</table>

Explain rationale for your rating:
8.11 **AACPX - Application Interface Complexity**

What are the nature of the interfaces between the main application system or subsystem and the glue code used to connect the system to the COTS/NDI components? Are there difficult synchronization issues? Must the interface balance conflicting criteria (e.g., security, safety, accuracy, ease of use, speed)?

*Metric: the same subjective averaging of the items in table VIII.A as used for the driver ACPWX. See the explanation provided for rating that driver under item 8.7, and then repeat the use of table VIII.A to evaluate this current driver.*

<table>
<thead>
<tr>
<th></th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point</td>
<td>Point total</td>
<td>Point total</td>
<td>Point total</td>
<td>Point total</td>
</tr>
<tr>
<td>total</td>
<td>is between</td>
<td>is between</td>
<td>is between</td>
<td>is between</td>
</tr>
<tr>
<td>5 and</td>
<td>8 and 10.</td>
<td>11 and 13.</td>
<td>14 and 15.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>N</th>
<th>H</th>
<th>VH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Explain rationale for your rating:
<table>
<thead>
<tr>
<th>Complexity Elements</th>
<th>Very Low (point value = 1)</th>
<th>Low (point value = 2)</th>
<th>Nominal (point value = 3)</th>
<th>High (point value = 4)</th>
<th>Very High (point value = 5)</th>
<th>Corresponding Points</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interface Conventions</strong> (e.g., naming, relevant usage scenarios, service signature, service order)</td>
<td>N/A</td>
<td>Nearly all API conventions are clear and consistent.</td>
<td>Most API conventions are clear and consistent.</td>
<td>Few API conventions are clear and consistent.</td>
<td>API conventions are non-existent.</td>
<td>———</td>
</tr>
<tr>
<td><strong>Control Aspects</strong> (e.g., consistent and clear error handling/recovery)</td>
<td>N/A</td>
<td>Nearly all control aspects are well defined and consistently applied.</td>
<td>Most control aspects are well defined and consistently applied.</td>
<td>Few control aspects are well defined and consistently applied.</td>
<td>No control aspects are well defined and consistently applied.</td>
<td>———</td>
</tr>
<tr>
<td><strong>Data</strong> (e.g., conversion, number/range typing)</td>
<td>No data conversion required.</td>
<td>Little data conversion required and standard data types used.</td>
<td>Some data conversion required and standard data types used.</td>
<td>Significant data conversion required and/or use of non-standard data types.</td>
<td>Extensive data conversion required and/or use of non-standard data types.</td>
<td>———</td>
</tr>
</tbody>
</table>

**Total Point Score** =

**Repeat of Table VIII.A for 8.11 calculation**
8.12 ACPER - Constraints on System/subsystem Technical Performance

How severe were/are the technical performance constraints (e.g., storage, memory, reserve, flow through capacity, etc.) on the application system or subsystem that the COTS/NDI components needed to/must meet?

*Metric: the presence or absence of constraints.*

**UNKNOWN**

<table>
<thead>
<tr>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>There are no technical constraints or real time processing needs.</td>
<td>Real time processing must be performed OR other technical constraints exist.</td>
<td>Real time processing must be performed AND other technical constraints exist.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>H</th>
<th>VH</th>
</tr>
</thead>
</table>

Explain rationale for your rating:
8.13 ASPRT - System Portability

What were/are the overall system or subsystem portability requirements that the COTS/NDI component needed to/must meet?

**Metric: the nature of portability requirements.**

<table>
<thead>
<tr>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>There are no portability requirements at the system/subsystem level.</td>
<td>System must be portable across platforms within the same family (e.g., across different versions of UNIX).</td>
<td>System must be portable across divergent platforms (e.g., from UNIX to VMS).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N</th>
<th>H</th>
<th>VH</th>
</tr>
</thead>
</table>

Explain rationale for your rating:
8.14 AAREN - Application Architectural Engineering

How adequate/sophisticated were the techniques used to define and validate the overall systems architecture?

*Metric: architecture validation techniques.*

### UNKNOWN

<table>
<thead>
<tr>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
</tr>
</thead>
<tbody>
<tr>
<td>No architecture validation done.</td>
<td>Paper analysis performed.</td>
<td>Peer reviews of architectural design (including interface definitions).</td>
<td>Prototyping/demos of the architecture performed.</td>
<td>Simulations of the architecture created.</td>
</tr>
</tbody>
</table>

![Ratings Scale]

Explain rationale for your rating:
IX. Application Effort Due to COTS Volatility Data

The Application Effort Due to COTS Volatility Model is defined in both approximate and detailed versions. The Approximate model determines added effort as simply a percentage of overall new application effort based on the percentage of code breakage due to COTS volatility as determined in question 7.18. The Detailed model adds the refinement of including the COCOMO II non-linear scale factors. (When used to perform an estimate, rather than ask for the actual new application effort, both models will assume an original Application development effort has been estimated using COCOMO II.)

9.1 **New application coding effort excluding** effort due to COTS integration, i.e., exclusive of glue code and tailoring:

   New application code effort: ________________ (person-months)

9.2 Percentage of **new application coding effort** due to breakage **excluding** breakage related to COTS integration, i.e., as a result of requirements change, but not due to design or programming error (this is the same breakage term as defined for COCOMO II):

   Application code breakage due to requirements change: __________%  

9.3 Percentage of **new application coding effort** due to breakage **as a result of COTS product volatility**, i.e., as a result of releases by the vendor of updated versions of COTS components.

   Application code breakage due to COTS volatility: __________%
9.4 COCOMO II Scale Factor Ratings:

Circle the appropriate box for each factor, one box per row:

<table>
<thead>
<tr>
<th>Scale Factor</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precedentedness</td>
<td>thoroughly unprecedented</td>
<td>Largely unprecedented</td>
<td>somewhat unprecedented</td>
<td>generally familiar</td>
<td>largely familiar</td>
<td>thoroughly familiar</td>
</tr>
<tr>
<td>Development</td>
<td>rigorous</td>
<td>Occasional Relaxation</td>
<td>some relaxation</td>
<td>general conformity</td>
<td>some conformity</td>
<td>general goals</td>
</tr>
<tr>
<td>Flexibility</td>
<td>little (20%)</td>
<td>some (40%)</td>
<td>often (60%)</td>
<td>generally (75%)</td>
<td>mostly (90%)</td>
<td>full (100%)</td>
</tr>
<tr>
<td>Architecture / Risk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resolution</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team Cohesion</td>
<td>some difficult interactions</td>
<td>Basically cooperative interactions</td>
<td>largely cooperative</td>
<td>highly cooperative</td>
<td>seamless interactions</td>
<td>N/A</td>
</tr>
<tr>
<td>Process Maturity</td>
<td>Chaos</td>
<td>CMM Level 1</td>
<td>CMM Level 2</td>
<td>CMM Level 3</td>
<td>CMM Level 4</td>
<td>CMM Level 5</td>
</tr>
</tbody>
</table>

* percentage of module interfaces specified, percentage of significant risks eliminated.
X. Free Form Comments

Please write down any comments or descriptions of items you feel are important to COTS integration but that were missing from this survey:

This concludes this survey. Thank you for your efforts. Your cooperation is greatly appreciated and goes a long way to ensuring the usefulness of the COCOTS software integration cost estimation tool.
Appendix C

Summaries of Collected COTS-based Systems Data
As of this writing, there are some 20 historical COTS-Based System development project data points in the COCOTS calibration database. These are the result of 200+ hours of on-site data collection interviews conducted with knowledgeable project personnel—usually a project manager or technical lead—from software developer organizations all across the United States. The information for each project includes records of actual expended effort and schedule, along with after-the-fact characterizations of the projects in terms of the various COCOTS submodel cost parameters discussed previously in the body of this thesis.

The data collection interviews noted above typically lasted a half-day, and were conducted by the thesis author and Dr. Elizabeth Clark (again, see the footnote on page 26 in chapter 2) working as a team. During the course of a typical interview, Dr. Clark and the author guided the interviewee through the COCOTS data collection form (see appendix B).

Follow-up interviews were later conducted as needed to clarify any discrepancies that arose in the data during the data conditioning and transcription process. These follow-up interviews were usually conducted over the phone, but on occasion there were opportunities for return site visits.

The data currently has a bias towards large aerospace and defense organizations, but the intention is that in the future attempts will be made to collect project data

* A Microsoft Access database was created to serve as an electronic repository for the COCOTS calibration data. Upon return from a site visit, data from the completed paper survey forms were transcribed into this database.
from a broader spectrum of software developer organizations, in particular the commercial business sector. A key feature of the existing data, however, is that much of it was reported in terms of specific kinds or “classes” of COTS products. After the first few data collection interviews, it was determined that asking questions in terms of classes of products was the most effective way of eliciting the information desired.

To alleviate concerns about the disclosure of proprietary information, confidentiality agreements were entered into between the author and the commercial organizations that contributed CBS project data to the COCOTS database. As a result, it is not possible to publish the raw project data that was collected during the course of this research. However, characteristics of the data in aggregate are shown throughout the rest of this appendix.

In terms of project domains, the data currently available is distributed as follows:

- Air Traffic Management (40%)
- Business (including databases) (15%)
- Comm/Navigatioj/Surveillance (20%)
- Logistics (5%)
- Mission Planning (5%)
- Operations (10%)
- Web-based Maps (5%)
The various classes of COTS products that were used by these projects include the following items:

- Back office/retail
- Configuration mgmt/build tools
- Databases
- Data conversion packages
- Device drivers
- Disk arrays
- Compilers
- Communication protocols/packages
- Emulators
- Engineering tools
- Graphic information systems
- GUIs/GUI builders
- Middleware
- Network managers
- Operating Systems
- Problem mgmt
- Report generators
- Software process tools
- Telecommunication & infrastructure
- Telemetry analysis
- Telemetry processing
- Word processing

There are varying amounts of data on each of the above items. Some classes are represented by several examples, many by just one or two. However, the number of distinct classes will likely grow if data collection continues in the future.
The development processes followed by the various projects while creating these COTS-based systems:

- Waterfall (40%)
- Spiral (35%)
- Incremental (20%)
- Evolution/Prototype (5%)

Overview of data findings

As stated above, the COCOTS calibration database currently contains information on some 20 industrial software projects. The hope is that the database will continue to grow, affording even greater insights into COTS-based systems development. Meanwhile, there are enough data points currently available to draw some preliminary conclusions. Keep in mind, though, that the statements that follow are at best rules of thumb gleaned from trends that seem apparent in the data—there are countervailing examples in each case.

All COTS assessment activity on a project appears most typically to be completed within 6 calendar months, with no more than 6 person-months in effort expended to do assessment for any given class of COTS components.

The spread in typical schedule for completion of the tailoring of all COTS components is greater than that for assessment, but still, more often than not, all tailoring appears to be completed within 6 calendar months—with again, no more than 6 person-months of effort expended for any given class of COTS products.
The data indicates Glue code typically takes longer and requires more effort to complete than tailoring. This may be because the “intellectual effort” required to simply configure (or tailor) a given COTS product is usually less than that required to create code around it that is not only new but also highly constrained—the situation that exists with glue code. The typical overall schedule for creation of all glue code in a system appears to be from 1 to 2 years, with up to 2 person-years of effort expended.

The effort associated with managing the impact of the volatility of COTS components on the larger system continues throughout the life of the project all the way through system retirement. However, the relatively low percentage of system rework due to COTS volatility reported for most of the projects in the existing database probably reflects the database’s current reporting horizon: the end of the initial development phase. If as intended the modeling scope of COCOTS is expanded in the future to cover the long-term operations & maintenance phase of a project, the relative percentage of system rework due to COTS volatility can be expected to increase.

Finally, note that within a given COTS component integration activity (assessment, tailoring, glue code writing, and managing COTS component volatility on the system as a whole), there is large variation in the data collected, as evidenced by the wide standard deviation intervals around the mean percentage of total COTS component integration effort represented by each activity. This illustrates the fact
that every CBS development is unique, making generalizations problematic. For instance, while some projects may follow a profile involving a certain amount of assessment activity, followed by some tailoring and glue code construction, other projects may follow different profiles involving almost nothing but tailoring activity with very little glue code being written, or vice versa. One thing that can be said, however, is that the data clearly show that certain activities tend to be more intensive depending upon the kind of COTS components being integrated into the COTS-based system. This knowledge can help CBS designers to better understand where their project development efforts are likely to be concentrated.
Following are summaries of the existing CBS project data in the form of histograms:

**Project delivery dates**

![Histogram of Project Delivery Dates]

**Overall project duration**

![Histogram of Overall Project Duration]
Overall project effort

Overall project SLOC
Project glue SLOC (reflects multi-COTS classes per project)

<table>
<thead>
<tr>
<th>SLOC</th>
<th>0-6k</th>
<th>6k-12k</th>
<th>20k-40k</th>
<th>60k-80k</th>
<th>100k-500k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>10</td>
<td>1</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Project COTS assessment activity duration

<table>
<thead>
<tr>
<th>Duration</th>
<th>0-0.5yrs</th>
<th>0.5-1yrs</th>
<th>1-2yrs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Project COTS tailoring activity duration (reflects multi-COTS classes per project)

Project COTS glue code creation duration (reflects multi-COTS classes per project)
Project COTS assessment activity effort (reflects multi-COTS classes per project)

![Graph showing effort distribution for COTS assessment activity.](image)

Project COTS tailoring activity effort (reflects multi-COTS classes per project)

![Graph showing effort distribution for COTS tailoring activity.](image)
Project COTS glue coding activity effort (reflects multi-COTS classes per project)

Project system effort due to COTS volatility
Project percentage system rework effort due to COTS volatility

Project total COTS effort in system (assessment+ tailoring+glue code+ volatility effort)
Mean % of Total COTS Effort by Activity (+/- 1 SD)

Median Initial Filtering Effort by COTS Class

- generic: 0.63
- telemetry processing: 2.00
- operating systems: 0.50
- network managers: 0.27
- GUls/GUI builders: 0.10
- emulators: 0.08
- databases: 1.91
- data conversion packages: 0.75
Appendix D

Guidelines for Using COCOTS with COCOMO II
COCOTS and its parent model, COCOMO II, have a very specific relationship. Understanding the nuances of this relationship, of exactly how the two models complement each other, is critical for anyone attempting to use the models to provide an integrated estimate of the effort and schedule that will likely be required to develop a COTS-based software system. Put another way, the need is to understand how estimates derived from the two models individually can be legitimately combined to arrive at a single unified estimate for a system being built out of a mix of new software, reused software, and commercial-off-the-shelf software components.

The key is to be clear about three things: 1) what the two models have been uniquely designed to estimate, 2) the exact phases of the software development life cycle that are covered by the two models, and 3) the relative fidelity of the two models.

What the Models Estimate

Let us begin with COCOMO II. Since its initial publication over 20 years ago in its original incarnation as COCOMO 81 all the way up to the current day, this model has always addressed two of the three avenues available for software system

*It is assumed that the reader of this appendix has a basic understanding of the details of both COCOTS and COCOMO II as presented in the body of the thesis to which this appendix is attached, including the definition of terms associated with these models. If the reader comes across terms or concepts which are unfamiliar, the reader should return to the main text of the thesis for clarification.
development: 1) writing new software from scratch, and 2) adapting or reusing software (specifically, source code) previously written for some other system.

The fundamental input parameter to the COCOMO effort model is an estimate in terms of either lines of code or function points of the amount of software that will either have to be written from scratch or adapted. The output of the model is an estimate in person-months of the amount of effort that will be required to write or adapt the specified amount of code.

COCOTS has been designed to address the third avenue available for software system development, the integration of black box off-the-shelf components. The phrase “black box” is important. COCOTS addresses the problem of developing effort and schedule estimates for those elements of a software system being constructed from software components for which no source code is available. This is precisely the root motivation for the development of COCOTS. Since COCOMO’s primary input parameter is software component size in lines of code or function points, it has no ability to calculate an effort estimate for the work involved with developing those parts of a software system that require the selection and integration of off-the-shelf components that supply functionality but for which no code (other than glue code or binding-ware) is actually being written or adapted. System development work of this nature can be significant, but COCOMO has no way to account for it—thus the need for COCOTS.
To account for development effort unassociated with the actual size of the component itself, COCOTS (as of this writing) uses three separate effort submodels compared to COCOMO's single effort model. The first is the assessment submodel, whose primary input is the number of off-the-shelf components being examined for possible inclusion in the COTS-based system under development and whose output is an estimate in person-months of the amount of effort that will be required to assess the specified number of components until one or more are selected for integration.

The second COCOTS effort submodel is the tailoring submodel. The primary input in this case is the number of off-the-shelf components being tailored or configured to work within the context of the software system into which they are being integrated, qualified by the complexity of the work needed to tailor a given class of components for that software system. The output is an estimate in person-months of the amount of effort that will be required to tailor the specified number of off-the-shelf components for use in the system being developed.

The third COCOTS effort submodel is the glue code submodel. The basic input parameter for this submodel is an estimate in terms of either lines of code or function points of the amount of software that will have to be written to either plug the various off-the-shelf components into the larger software system or to get them to talk to each other. The output of the submodel is an estimate in person-months of the amount of effort that will be required to write the specified amount of glue code.
But things don’t end there, with COCOMO II handling new and reused code and COCOTS handling off-the-shelf components. To decide which model to use to generate a given estimate for the various parts of a COTS-based system, it is necessary to also look closely at both the nature and intended use of any COTS components under consideration.

To begin, it must be understood that COTS products can be used in three ways: 1) as a component of a tool bed, 2) as a component of a system development or support infrastructure, and 3) as a component of a new application.

For the first two items, COTS as tools and COTS as infrastructure, COCOMO II is still the primary model to use when trying to capture the cost impact of using these kinds of off-the-shelf components in the development and support environments within which a COTS-based system is being designed and constructed. Specifically, the contribution to overall system development effort deriving from the use of COTS components as part of a tool bed or as infrastructure can be addressed by COCOMO II via the following drivers (see section 2.4.1 in the main body of this thesis):

- **COTS as tools**: Use of Software Tools (TOOL) and developer Language & Tool Experience (LTEX).

- **COTS as infrastructure**: Platform Volatility (PVOL) and developer Platform Experience (PLEX).
Again, however, it must be understood that using these two COCOMO II drivers to model the impact of COTS components used as tools or infrastructure still does not quite capture everything. What is being missed are any associated licensing or assessment costs. For the latter, if they are expected to be measurable, the COCOTS assessment submodel should be used.

Now what of the last use for COTS components indicated above, COTS as an application component? This is where the use of COCOTS comes to the fore, but again, only after an examination of the off-the-shelf component under consideration. If the item is truly a black box application component that comes with no access to its original source code, then COCOTS is the estimation model to use. This is the exact situation for which COCOTS was designed.

However, not all off-the-shelf components are acquired as black box items, sometimes they do in fact come supplied as uncompiled source code (i.e., "white box" components). In this situation—unless the COTS source code is simply compiled unexamined directly "as is" along with the rest of the original system source code—the off-the-shelf code should be viewed in this case as adapted or reused software and the COCOMO II reuse model should be used to estimate its associated integration effort.
Figure D.1 illustrates the appropriate use of the estimation models as discussed above.

**Figure D.1** – COTS-based system components mapped to appropriate estimation model.

Figure D.2 illustrates a hypothetical COTS-based system architecture, highlighting the various possible components in such a system. Compare figures D.1 and D.2 to see which estimation models should be applied to the various kinds of elements in figure D.2.
KEY
NEW – new code component; REUSED – reuse component; COTS – COTS component;
COTS WB – white box COTS; COTS BB – black box COTS; COTS TOOLS – tool component

Figure D.2 – a hypothetical COTS-based system architecture.
Aside from the costs of assessment, tailoring and glue code, there is one other cost associated with the use of COTS components that both COCOTS and COCOMO II try to address. This is the cost of effort due to any volatility in the COTS components. (Volatility in this context refers to the frequency with which a COTS vendor releases new versions of its product along with the degree of change that occurs within each of those releases.)

Within COCOMO II, the impact of COTS volatility on the new or reused software components is captured via the REVL term in the COCOMO II sizing equations. This inflates the effective size of the software being written or adapted based upon the estimated change in that software that will be required prior to initial acceptance testing of the COTS-based system due to system requirements evolution and COTS product volatility. A similar term called CREVOL appearing in the glue code submodel of COCOTS captures the impact of those effects on the glue code being written to bind the various COTS components to the COTS-based system.

(Early versions of COCOTS had a separate submodel addressing COTS volatility effects on the larger system but this was eventually eliminated as being redundant once the REVL term in COCOMO II was redefined to include COTS volatility effects.)

Finally, a word of caution about certain costs COCOTS may not yet adequately capture—that is, costs it may still have a tendency to underestimate. The first is overall COTS component system integration work. In particular, the glue code
submodel captures some of this work in its glue code sizing equation, but what it does not necessarily reflect is the shear number of COTS components for which glue code is being written. For example, there is probably more effort associated with designing, coding and testing 50,000 lines of glue code written for ten off-the-shelf components than 50,000 lines of glue code of similar complexity but written for say only three COTS components. As currently formulated, the glue code model will not capture that differential.

Similarly, with regard to capturing COTS volatility effects, it may be that these are also not being adequately captured by COCOTS when it comes to their impact on COTS component tailoring. Again, as currently formulated, there is no term within the tailoring submodel that directly addresses the impact of COTS volatility on the COTS tailoring activity. However, that being said, it may be that these effects can indeed be captured by the tailoring submodel as currently formulated by a slight redefinition of the Tailoring Complexity Qualifier (TCQ) term already present in the model to account for the anticipated impact of any COTS volatility.

The Phases of the Development Life Cycle the Models Cover

Let us begin with COCOMO II. This model is very specific in the activities and/or phases in the software development life cycle that are directly covered by the estimates that the model produces. COCOMO II directly estimates those activities that begin right after the traditional Software Requirements Review (SRR) milestone
and continuing up through integration and test and the Software Acceptance Review (SAR) milestone. The important point here is that COCOMO II does not directly capture any of the requirements definition activity.

COCOTS, on the other hand, *does* capture some of the requirements definition activity in its Assessment submodel. This is due to the fact that adopting the COTS approach to building a software system forces at least some system requirements definition and COTS product selection to be done in tandem, because final system requirements ultimately will have to bend to whatever functionality is offered by the COTS products currently on the market.

The result is a slight mismatch in the activities covered by COCOMO II and those covered by COCOTS as illustrated in figure D.3. The consequences of this only become an issue if the COTS product assessment activities associated with a given COTS-based system development end up representing a significant portion of the overall system development effort. In that case, be cautioned that if you adhere to the traditional COCOMO II coverage endpoints as the convention for reporting your overall COTS-based system development effort estimate as derived from a combined COCOMO II-COCOTS modeling effort, you may be somewhat *over* estimating the effort that will be required for the development activities that occur between the traditional SRR and SAR milestones.
The Relative Fidelity of the Two Models

The critical thing to keep in mind is that COCOMO II is an operational model while COCOTS is still very much experimental. In particular, COCOMO II has been calibrated to *eight* times the number of historical software project data points that
were available (as of this writing) to calibrate COCOTS. In some respects, this means that comparing estimates from the two models is like comparing apples and oranges, they’re just not the same thing. However, as a practical matter, a reasonable assumption about the “goodness” of a system development effort estimate based upon the combined effort estimates from both models might be viewed as a function of the relative contribution each model makes to the overall development estimate in any given instance. The unstratified and cross-validated predictive fidelity of COCOMO II calibrated to 161 data points has been demonstrated to be within (+/-) 30% of effort actuals about 70% of the time. On average across the three COCOTS submodels, the corresponding measures for COCOTS based upon 20 calibration data points also unstratified but lacking cross-validation are (+/-) 30% of effort actuals 60% of the time.

Though the model fidelity numbers are similar, the law of averages as well as just plain common sense suggest that your belief in the accuracy of a COCOMO II estimate should be much stronger than your corresponding belief in a COCOTS estimate because the COCOMO II fidelity numbers are based on a much larger calibration dataset. But how might you quantify your relative faith in the two models?

Because COCOMO II and COCOTS are deterministic models (i.e., given the exact same input parameters, the models will return the exact same output estimates every time), it is not possible to define a variance-based, probabilistic confidence
interval around the point estimates produced by these models. Some other measure must be used to characterize their relative fidelities.

One approach might be to recognize that your faith in a combined COCOMO II/COCOTS estimate might reasonably increase or decrease in strength along some scale proportional to the relative contribution of each model to the overall effort estimate. The more your final estimate is based upon COCOMO II, the stronger your belief in that estimate should be; the more your final estimate is based upon COCOTS, the opposite might be true.

Based upon the eight-to-one ratio in the relative sizes of the databases used to calibrate the two models, I am proposing that a metric called "confidence number" be used to characterize the relative fidelity of a combined COCOMO II/COCOTS, COTS-based system development effort estimate:

$$\text{Relative } \text{CN} = \left[ (\text{CNcoc} \times \text{ESTcoc} / (\text{ESTcoc} + \text{ESTcots})) \right] + \left[ (\text{CNcots} \times \text{ESTcots} / (\text{ESTcoc} + \text{ESTcots})) \right]$$  
Eq. D.1

where

Relative CN = "confidence number" for a given combined estimate.

CNcoc = "confidence number" for COCOMO II defined = 8.

CNcots = "confidence number" for COCOTS defined = 1.

ESTcoc = portion of total COTS-based system development effort provided by COCOMO II (in person-months).
ESTcots = portion of total COTS-based system development effort provided by COCOTS (in person-months).

Thus a relative degree of belief in a combined COCOMO II/COCOTS effort estimate can be assigned a number along a sliding scale from 1 to 8 where 8 represents your best possible confidence in the estimate corresponding to a pure COCOMO II estimate and 1 represents a more experimental confidence in the estimate corresponding to a pure COCOTS estimate.

As the size of the COCOTS calibration database grows in the future, the appropriate values assigned to CNcoc and CNcots can be revisited, until ideally the two numbers will reach parity and this relative “confidence number” metric will no longer be needed.

A Procedure for Applying the Models Together

1) Determine which software elements of your COTS-based system will be developed using new code, adapted code, and black box COTS products.

2) Use COCOMO II to generate effort estimates for the new code and adapted code portions of the system.

3) Use COCOTS to generate effort estimates for those portions of the system using black box COTS products.
4) For the sake of simplicity, adopt the traditional COCOMO II estimation coverage endpoints and add the COCOMO II and COCOTS effort estimates together to arrive at the total COTS-based software system development effort estimate; however, keep in mind the caveats discussed previously regarding what can happen to your effort estimate if COTS assessment effort represents a significant portion of your overall system development effort.

5) Determine the relative “confidence number” for your overall estimate as defined in equation D.1.

6) To determine a total schedule estimate, enter your combined effort estimate into the existing COCOMO II schedule equation. (It is recognized that more investigation needs to be done regarding the most appropriate model parameters for a combined COCOMO II/COCOTS schedule model, but as of this writing the most viable approach is to use the current COCOMO II schedule model.)

7) END OF PROCEDURE.