

# 19. SOFTWARE PRICING

## Table of Contents

<b>19.1 INTRODUCTION.....</b>	<b>19-2</b>
<b>19.2 SOFTWARE PRICE ANALYSIS.....</b>	<b>19-2</b>
19.2.1 Price Analysis Techniques.....	19-5
19.2.2 Price Analysis Tools.....	19-6
<b>19.3 SOFTWARE COST ANALYSIS.....</b>	<b>19-8</b>
19.3.1 Considerations For Software Cost Analysis.....	19-11
19.3.2 Cost Analysis of a Parametric Estimate.....	19-14
19.3.3 COTS Special Pricing Considerations.....	19-16
<b>19.4 SOFTWARE PRICING SUMMARY.....</b>	<b>19-16</b>
<b>APPENDIX 19A. SOFTWARE COST ESTIMATION</b>	
<b>TERMINOLOGY.....</b>	<b>19A-1</b>
<b>APPENDIX 19B. POPULAR PARAMETRIC SOFTWARE</b>	
<b>COST MODELS.....</b>	<b>19B-1</b>
<b>APPENDIX 19C. COMMERCIAL OFF-THE-SHELF (COTS)</b>	
<b>SOFTWARE.....</b>	<b>19C-1</b>
<b>APPENDIX 19D. WORKS CITED.....</b>	<b>19D-1</b>

## 19.1 INTRODUCTION

Software is a set of programs and accompanying documentation that directs computers to perform desired functions. In simple terms, a software program is a set of instructions for a computer.

## 19.2 SOFTWARE PRICE ANALYSIS

In order to conduct software price analysis, the terms for comparing the data must be properly defined. Referring to the Naval Center for Cost Analysis (NCCA) Software Development Estimating Handbook, Phase One, dated February 1998, there are two basic types of information required to develop a good software price analysis: 1) technical and programmatic information for the program proposal being analyzed and 2) technical and programmatic information for the analogous/similar historical programs used to develop the estimate/IGCE that will be compared to the proposal.

Based on the analysis in NCCA's Handbook, a software development estimate requires, at a minimum, the following information for the estimates being analyzed:

- Some measure of the work to be performed with associated units (i.e., source line of code (SLOC) counts, words, function points, etc.).
- If SLOC is utilized as the unit of measure, the associated counting convention (i.e., physical, physical with comments, logical, etc.).
- The condition of the code (i.e., percent new, percent reused (modified, verbatim, translated, rehosted, Commercial Off the Shelf (COTS), etc.)), with associated definitions.
- The phases of the software development life cycle to be estimated (e.g., System Design Review (SDR) through Formal Qualification Test (FQT)).
- The development mode (such as, embedded versus non-embedded). Embedded Software is defined as software that is inside a physical object and controls its behavior. This is a more or less specialized term for software inside navigational devices, radar sets, oscilloscopes, and other instruments. Embedded software has its own characteristic productivity and quality profiles. (Source: Capers Jones). This is software that determines the functionality of microprocessors and other programmable devices that are used to control electronic, electrical and electro-mechanical equipment and sub-systems. The programmable devices are often "invisible" to the user. (Source: The TickIT Guide.)

- Especially for historical costs, if known, the name of the contractor responsible for developing the program. NCCA contends that contractor-specific data holds the greatest possibility for increasing the accuracy and decreasing the variance associated with software estimating tools.

In addition to the above items, there are four other areas that can be cost drivers for software that are input parameters for most software parametric cost models:

- 1.) Project Application - MIS, communications, radar, etc.
- 2.) Specification Level - MIL-STD-2167A, commercial, etc.
- 3.) Development Model - Waterfall, Spiral, etc.
- 4.) Project Scope - rehost, development, maintenance, etc.

Since all of the information requested affects the projected productivity of the development effort, it is crucial that the information gathered be as specific as possible. In addition to the aforementioned information on the program being estimated, the analyst must compile the same information for the analogous historical programs that will be used to develop the price analysis comparison. Furthermore, the actual effort, schedule, and cost (price) to develop the software, by software development phase if possible, should be obtained. With this information the most accurate productivity, schedule, and labor rate metrics can be developed. The Software Development Plan (SDP) typically requires a list of previously delivered programs developed by the contractor, with the associated technical and programmatic data. If, however, the SDP is not available, this type of information can and should still be obtained from the contractor in whatever form is available. When collecting historical data, the analyst must ensure that the information is for completed programs because projections of on-going efforts are often mixed in with the list of the contractor's programs. Since software development is continuously evolving, the analyst should always try to obtain the most recent data available.

It is important to know how the SLOC was counted so that any productivity or effort estimating relationships developed will be valid. There are two main categories of code counting conventions: physical and logical. Counting physical SLOC is accomplished by tallying the number of carriage returns in the source document. Logical SLOC are counted by tallying logical units (for example, an IF-THEN-ELSE statement is considered one logical unit).

The impact of the code counting convention is emphasized in the NCCA Handbook, Phase One, which referenced two studies. An Institute for Defense Analyses (IDA) study found that on average, physical SLOC produce a code size that is about 20 percent higher than counting the same code using a logical SLOC definition. NASA's Software Engineering Laboratory (SEL) also found wide differences between physical and logical code counts. They found that a FORTRAN program's ratio of physical lines to logical statements ranged from 2.5 to 5 due to variations in the number of comments. Likewise, Ada programs exhibited a similar ratio of 2.5 to 6 physical lines per logical statement. Not only is knowing the amount of source code necessary, but knowing the "condition" of the code is also important. NCCA used the term "condition" to describe the composition of the source code (i.e., %new, %reused). Sections 19.2.4 (Technology Insertion) and 19.4.3 (Reused Code) contain additional information concerning reused code. The amount of high-order language (HOL) a program contains is also an important factor to consider. All programming languages, except Assembly, are defined as HOLs. Analysts should ask for the new and reused SLOC by language so as to avoid having to derive these values.

When using historical software effort data, it is important to consider the level of requirements under which the software was developed. A major program may have several software development efforts spanning different acquisition phases. For example, typical acquisition strategies require development of prototypes and associated software during a Prototype Phase. After a competitive selection process, one contractor's design is chosen for further development. Final development takes place for the deployable software by the winning contractor. The contractor probably reused code from the Prototype Phase that may not have undergone the same level of documentation, testing, or review as software developed for deployment. As a result, using historical prototype data points to estimate effort prior to deployment may not be appropriate without some adjustment.



Therefore, it is recommended that the data shown in Table 19-1 be requested from the offerors (if not already in the SIR) and from the sources of other data/IGCE to allow a valid comparison.

**Table 19-1. Software Estimate Comparison Parameters**

<b>Data Area</b>	<b>Description</b>
Project Application	What is the software's functional purpose
Project Size	Most commonly in SLOC but could be in Function Points or Object Points
Size Counting Convention	Defines what is included in the SLOC/FP/OP count
Project SW Language	If coded in more than one language, try to get percentages. As a minimum, know Assembly versus HOL
Project Phasing	What phases of the SW development schedule were included in the contract price
Code Condition	% new, % modified (need definition e.g., less than 20% recoded), % unmodified, % COTS
Development Mode	Embedded versus non-embedded as a minimum
Specification Type	MIL-STD-2167A, commercial, etc.
Development Model	Waterfall, Spiral, etc.
Project Scope	Rehost, development, maintenance, etc.

If as a minimum the information in Table 19-1 is obtained for the proposed project and the item(s) to be used as the basis of comparison (IGCE, market survey, similar items, etc.), then there should be good substantiation for the comparison.

### 19.2.1 Price Analysis Techniques

The FAA Procurement Guidance T.3.2.3A.1.c provides the following pricing techniques to use when performing price analysis:

- Comparison of proposed prices with Independent Government Cost Estimate.
- Comparison of proposed prices received in response to the Screening Information Request (SIR).
- Comparison of prior proposed prices and contract prices with current proposed prices for the same or similar end items and services in comparable quantities.
- Comparison with competitive published catalogs or lists, published market prices or commodities, similar indexes, and discount or rebate arrangements.

- Application of Software Cost Estimating Model parameters or Rules of Thumb (such as person-months per SLOC, or other units; see section 19.3.) (AMS uses the term “rough yardsticks” for this technique) to highlight significant inconsistencies that warrant additional pricing inquiry.
- Ascertaining that the price is set by law or regulation.

### 19.2.2 Price Analysis Tools

Price analysis tools for software are the same as those cited in Chapter 5. Some additional guidance applicable, not only to software analysis, but to all price analysis is provided below.

#### Comparison with a Similar Item's Proposal/Price/Cost Estimate

This section is based on a U.S. Air Force Material Command (AFMC) white paper “Methods for Evaluating Similar Items”, dated January 1996. The ability to compare a proposal with a similar item for price analysis presumes the price for that similar item is reasonable and acceptable. The Government might have purchased the item previously on the basis of adequate price competition, catalog or market pricing, commercial item pricing, or negotiations using cost or pricing data. If so, documentation that demonstrates price reasonableness for the item would already be possessed by the Government. Any of these should generate the confidence necessary to support the contention that price analysis will produce a reasonable price. However, it is also possible that the Government may never have purchased the similar item. In that situation, it would be necessary to establish the reasonableness of a similar item's price before any further price analysis on the offered item could be conducted. It is imperative to determine a suitable basis, else the price analysis would be without merit.

Once comfortable with the reasonableness of the similar item's price, an understanding of the technical similarities and differences between the offered item and the similar item is necessary. The pertinent characteristics (e.g., size, language, application type, etc., from Table 19-1) of each item must be identified to facilitate comparison.

If a direct comparison is not possible, break down the offered and similar items until a common baseline is reached. It may be as simple as segregating options and upgrades to the same basic, lower-level unit, along with two lists of adders that would complete the items. The automobile industry is the clearest example where this method can be applied. However, the same thing can be done with software, isolating differences such as functionality to leave the same basic operating system and application type. Then, proceed with establishing prices for the baseline unit and each adder.

The most complex, and perhaps most frequent, situation to be encountered, especially for a new development, involves the inability to reach a common, identifiable baseline unit. In this case compare the characteristics of the two items and determine some relationship between them. Examples could be found where similar items might be compared in areas such as size, language, development phases, and specification level. The price of the similar item, having been determined reasonable, is used as the baseline, and the differences between the two items are considered as pluses and minuses to that baseline.

When the procured and similar items are broken down for evaluation, any suitable price analysis tools and techniques may be used to substantiate prices of the segregated pieces. Utilize purchase history or catalog, market, or commercial price assessment as applicable. By closely aligning characteristics into comparable categories, some parametric relationship might be disclosed to explain the impact of a characteristic. Of course, if reasonableness of a portion cannot be established using price analysis techniques, the negotiator should request cost information on that portion and perform a cost analysis.

### The Cost/Price Model

The analyst should plan for the development of a cost/price model. This is not a Cost Estimation Model as discussed in Appendix 19B, but a spreadsheet type model used specifically for cost/price analysis and/or proposal evaluation. When preparing a negotiation position, a cost/price model should consist of spreadsheets for the basic items being negotiated (usually CLINs or WBS elements), summary sheets, and sheets containing backup data (other direct costs, rates etc.). Essentially, all of the elements that are to be analyzed should be represented in the model.

In a competitive procurement it is often useful to provide an automated model with the SIR for direct input by the contractors. This reduces evaluation time and minimizes errors. The construction of the model will depend heavily on the type(s) of contract, CLIN structure, WBS structure, and quantity of data required. The analyst must work closely with the CO and program office, to assure that mutual goals are met. A technical description and an in-depth explanation of cost/price models are provided in Chapter 3, "Automated Cost Models" of this Handbook.

For software procurements, the model can be especially useful since it should contain the input parameter requirements (Table 19-1 as a minimum) needed to compare proposals to the IGCE and/or similar items, plus be used as a source of data to be input into a Software Cost Estimation Model.

### 19.3 SOFTWARE COST ANALYSIS

Cost analysis is the review and evaluation of the separate cost elements and proposed profit/fee of:

- An offeror's or contractor's cost or pricing data or information other than cost or pricing data and
- The judgmental factors applied in projecting from the data to the estimated costs.

The purpose of the evaluation is to form an opinion on the degree to which the proposed costs represent what the cost of the contract should be, assuming reasonable economy and efficiency. However, cost analysis does not necessarily provide a picture of what the market is willing to pay for the product involved. For that price analysis is needed.

The cost areas analyzed in a Cost Analysis, each discussed in detail in a separate chapter of the FAA Pricing Handbook, are as follows:

- Direct Labor
- Material
- Other Direct Costs (ODCs)
- Indirect Costs
- Facilities Capital Cost of Money
- Profit/Fee

#### Cost Estimation: Approach and Methods

As excerpted from the 05/30/03 version of the Jet Propulsion Laboratory's Handbook for Software Cost Estimation, the following should be included in a proposed software estimate.

The dominant cost in software development is the cost of labor. A basic cost equation for software can be defined as:

$$Total\_SW\_Project\$ = SW\_Development\_Labor\$ + Other\_Labor\$ + Nonlabor\$$$

*SW\_Development\_Labor\$ includes:*

- Software Systems Engineering - performed by the software architect, software system engineer, and subsystem engineer for functional design, software requirements, and interface specification. Labor for data systems engineering, which is often forgotten, should also be considered. This includes science product definition and data management,



- Software Engineering - performed by the cognizant engineer and developers to unit design, develop code, unit test, and integrate software components, and
- Software Test Engineering - covers test engineering activities from writing test plans and procedures to performing any level of test above unit testing.

*Other\_Labor\$* includes:

- Software management and support - performed by the project element manager (PEM), software manager, technical lead, and system administration to plan and direct the software project and software configuration management,
- Test-bed development,
- Development Environment Support,
- Software system-level test support, including development and simulation software,
- Assembly, Test, & Launch Operations (ATLO) support for flight projects,
- Administration and Support Costs, including Overhead and G&A,
- Software Quality Assurance,
- Independent Verification & Validation (IV&V), and
- Other review or support charges.

*Nonlabor\$* includes:

- Support and services, such as workstations, test-bed boards & simulators, ground support equipment, network and phone charges, etc.,
- Software procurements such as development environment, compilers, licenses, CM tools, test tools, and development tools,
- Travel and trips related to customer reviews and interfaces, vendor visits, plus attendance at project-related conferences, and
- Training.

All estimates are made based upon some form of analogy: Historical Analogy, Expert Judgment, Models, and "Rules-of-Thumb." The role these methods

play in generating an estimate depends upon where one is in the overall life-cycle.

Typically, estimates are made using a combination of these four methods. Model-based estimates along with high-level analogies are the principal source of estimates in early conceptual stages. As a project matures and the requirements and design are better understood, analogy estimates based upon more detailed functional decompositions become the primary method of estimation, with model-based estimates used as a means of estimate validation or as a “sanity-check.”

1. Historical analogy estimation methods are based upon using the software size, effort, or cost of a comparable project from the past. When the term “analogy” is used in this document, it will mean that the comparison is made using measures or data that has been recorded from completed software projects. Analogical estimates can be made at high levels using total software project size and/or cost for individual Work Breakdown Structure (WBS) categories in the process of developing the main software cost estimate. High-level analogies are used for estimate validation or in the very early stages of the life-cycle. Generally, it is necessary to adjust the size or cost of the historical project, as there is rarely a perfect analogy. This is especially true for high-level analogies.
2. Expert judgment estimates are made by the estimator based upon what he or she remembers it took previous similar projects to complete or how big they were. This is typically a subjective estimate based upon what the estimator remembers from previous projects and gets modified mentally as deemed appropriate. It has been found that expert judgment can be relatively accurate if the estimator has significant recent experience in both the software domain of the planned project, as well as the estimation process itself [Hihn and Habib-agahi, 1990].
3. Model-based estimates are estimates made using mathematical relationships or parametric cost models. Parametric cost models are empirical relationships derived by using statistical techniques applied to data from previous projects. Software cost models provide estimates of effort, cost, and schedule.
4. “Rules-of-thumb” come in a variety of forms and can be a way of expressing estimates as a simple mathematical relationship (e.g.  $\text{Effort} = \text{Lines\_of\_Code} / 10$ ) or as percentage allocations of effort

over activities or phases based upon historical data (e.g. I&T is 22% of Total Effort).

Whatever method is used, it is most important that the assumptions and formulas are documented to enable more thorough review and to make it easier to revise estimates at future dates when assumptions may need to be revised. All four methods are used during the software life-cycle. The level of granularity varies depending on what information is available. At lower-levels of the WBS, expert judgment is the primary method used, while model-based estimates are more common at higher levels of the WBS.

### 19.3.1 Considerations for Software Cost Analysis

In addition to general cost analysis considerations, there are three key considerations that apply to most proposals for a software program. These are the software development productivity, code condition (percent new and reused code) and the software defect rate by program phase.

#### Productivity

Productivity is a primary indicator of how efficient the contractor is in developing software. Productivity relates software development effort to the organizational capabilities, experience, and individual talents of the team that will perform the software development. This rate should be based upon historical data. The productivity rate tends to remain constant for a given organization, so previous proposals by the same contractor are especially valuable. It is much more difficult to compare rates between different contractors because it is so dependent upon organization and personnel. Productivity is also affected by programming language, processes, specification level and software tools.

Referring to NCCA's Handbook, two calculations should be made for software development productivity:

1. Productivity expressed as hours per SLOC based on the following formula:

$$\text{Hours/SLOC} = \frac{\text{Total PM} * 152 \text{ Hours/PM}}{\text{Total SLOC}}$$

2. Productivity expressed as hours per new SLOC. This metric is important because new SLOC tends to drive the effort.

$$\text{Hours/New SLOC} = \frac{\text{Total PM} * 152 \text{ Hours/PM}}{\text{Total SLOC} * \% \text{ New SLOC}}$$

*PM = person-months of effort, 152 is assumed to be the average work-hours in a month.*

This rate along with the code size usually forms the basis for the software development effort. Therefore, it is important to compare productivity against a Rule of Thumb estimate, other programs and/or the contractors past performance to determine if it reasonable. Then, over the life of the contract, compare the rate from proposal to proposal. The rate does tend to increase (become less efficient) gradually over the life of the contract as additional requirements are made to the program (new code has to be integrated with more and more old code).

### Code Condition

Code condition is important to cost analysis. The proposal should separate the total software development effort by percent of new code, modified code, COTS and unmodified code.

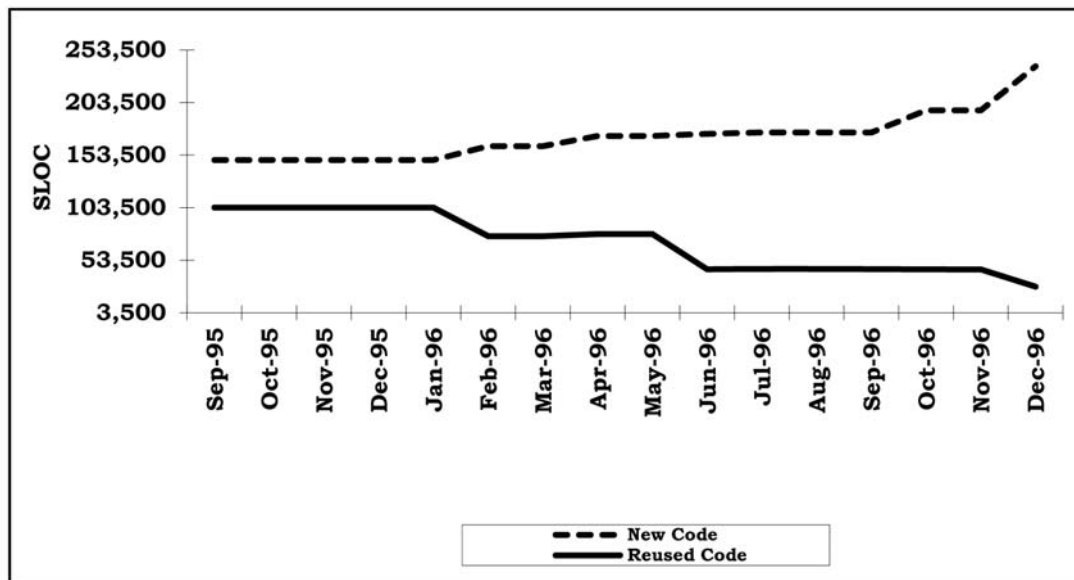
Reused code (modified, unmodified and COTS fall into this category) is included in a program to reduce effort, cost and time (schedule). According to NASA's Software Engineering Lab (SEL), "Cost and Schedule Estimation Study Report", dated Nov 1993, for projects with moderate to low code reuse (less than 70 percent), the post-CDR growth in SLOC due to requirement changes and items to be determined (TBDs) is commensurate with past SEL experience: 40 percent. For projects with high code reuse (70 percent or more), the post-CDR growth in SLOC is only about half as much (20 percent). For projects with moderate to low code reuse, the post-critical design review (post-CDR) growth in schedule is 35 percent. For projects with high reuse, the post-CDR growth in schedule is 5 percent.

Once the percentage of modifications in a block of code exceed 20%, it is usually less expensive to write new code. Often contractors are optimistic in the amount of code that can be reused. Based on metrics of software program growth (increasing cost and schedule), there are typically two sources of new code increase, requirements growth from the user and less modified/unmodified code than projected by the contractor. As the contractor gets into the process of actually developing the software, the task of reusing software can become more difficult than planned. Therefore it is important to keep track of these percentages over the life of a contract as both indicators of future problems and for cost analysis.



Figure 19-1 below is an example of new code growth at the expense of reused code taken from an actual FAA program. Although total SLOC grew only 5% during the 17 month period shown, the amount of new code that had to be developed grew from 59% of the total code to 89%. There was also about 25% schedule growth on this program.

Figure 19-1. Comparison of New Code versus Reused Code



### Software Defects

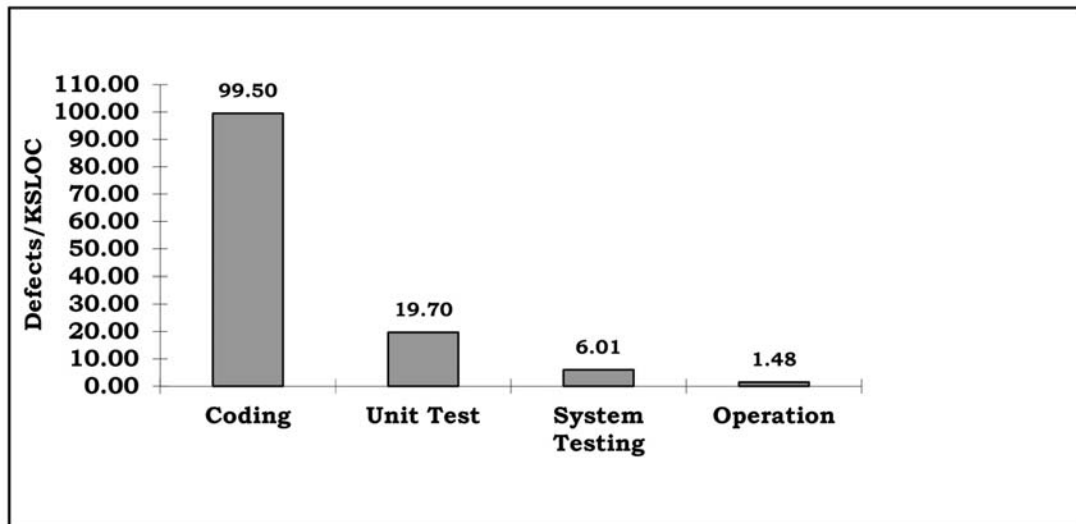
The contractor should indicate in the proposal the number of defects expected to be discovered and fixed for each phase of the program. The earlier in the development cycle that a software bug is discovered/detected, the less expensive it will be to fix. The quantity of software defects should be expressed in or converted to a rate. A common defect rate formula follows (assuming lines of code are counted).

$$\text{SW Defect Rate} = \frac{\text{Defects}}{\text{SLOC}}$$

There should be separate rates calculated by phase using the SW Defect Rate equation for new code and modified code, plus a rate for defects remaining in the existing, unmodified code. These calculations would then become the basis for the effort required for testing/engineering to find and to fix the defects in each phase. Defect metrics can be useful during software cost analysis to aid the analyst in determining if the contractor's estimated effort for identifying and fixing defects is consistent with the contractor's/industry's historical averages.

Figure 19-2 shows a typical trend for detecting software defects based on studies of completed software programs by Musa, Iannino and Okumoto in their book *Software Reliability-Measurement, Prediction, Application*. The defects discovered during coding and unit test are usually not formally reported/tracked by the contractor so expect that only the estimates for the phases after Unit Test will be provided in a proposal. Musa et. al., per their study, organized the data to predict the number of defects remaining in the software per KSLOC at the start of each phase. Each defect quantity includes the defects for the remaining phases (6.01 is included in the 19.70). The book indicated that program size (SLOC) was the most important defect prediction factor, with specification change activity, programmer skill level and design documentation thoroughness being the other most significant predictive factors. Defects continue to exist in the software even after site installations are completed that are gradually discovered as changes in software use or hardware occur.

Figure 19-2. Mean Software Defect Rate by Phase



### 19.3.2 Cost Analysis of a Parametric Estimate

Since most software cost estimates and proposal estimates today are based at least in part on parametric models, this section summarizes some points to consider when analyzing a parametric estimate. This section is extended from NASA's Parametric Cost Estimating Handbook, dated fall 1995.

With the proliferation of parametric cost estimating models and tools, both commercial models and "home grown" versions, it is impossible to describe what to look for in every model and cost parameter. However, some generalizations can be made. An analyst confronted with a parametric cost estimate should take a few steps to ensure a fair review. These are:

- Understand the cost model used. Appendix 19B and 19C can be of assistance.
- Review the program inputs to the model. Is the schedule correct? Does the WBS adequately describe the product being estimated? Is anything missing? Are there duplications? Does the WBS follow the statement of work?
- Review the technical inputs to the model with government engineers or program office. Check them for reasonableness and benchmark them using the experience of the resident experts.
- Understand the model's cost drivers. Generally, there are a few select parameters that are the predominant cost estimating factors that drive cost. Many of the others are just "tweaks" to the model. Concentrate on the cost drivers when performing the analysis.
- Be aware of the assumptions the cost estimator made when the model was built. Are they still reasonable for this procurement?
- Be knowledgeable of the historical cost basis for the model, if any. Be sure to review the source documentation. Be wary of any model used that has no basis in historical cost. How was the data "tuned" or normalized? Were data outliers disregarded? If so, why? Was the model calibrated? How was the calibration performed? Were any universal or "standard" cost factors used? Would they be applicable in this case?
- Question how the future environment might be different from the historical case. Have these differences been accounted for?
- Review the track record of the estimator or the estimating organization. What is their past performance? Have their estimates been reliable?
- Understand the labor factors involved with the model. What are the effective costing rates used? Are they reasonable? Do they reflect the organization and skill levels being estimated? Are the skill levels appropriate for the type of work being performed?
- Identify what cost factors have been "tweaked" and why. Focus on the "big ticket" items. Using expert opinion and "rules of thumb," are any significant cost factors outside the range of reasonableness? For instance, it is very easy to calibrate a software cost model's cost (hours or dollars) per line of software code. Is the cost estimating relationship (CER) reasonable for the estimate? Since models

may define a software line of code differently, it is important to understand the definitions used in the model being evaluated.

### 19.3.3 COTS Special Pricing Considerations

See Appendix 19C for a detailed discussion of the areas that are special and should be considered in pricing a proposal that is for or includes COTS items.

## 19.4 SOFTWARE PRICING SUMMARY

Pricing is used by procuring officials to establish a “fair and reasonable price”. Price analysis and cost analysis are the two basic techniques used to accomplish this purpose. Price analysis should always be performed. Under certain circumstances, cost analysis also needs to be performed. Price analysis plays the lead role in determining price reasonableness and fairness, and it becomes the responsibility of the analyst to research and gather pricing data from sources other than the contractor. As a result, the analyst must understand the fundamentals of performing price analysis along with the peculiar aspects associated with software cost estimation. In addition to the general cost analysis considerations, there are three key considerations that apply to most proposals for a software program: software development productivity, code condition (percent new, reused code) and the software defect rate by program phase. The software estimation and analysis techniques/methods contained in this chapter should provide the analyst with the background required to understand a software program cost estimate and, therefore, aid in analysis of the proposed estimate.

Software estimating models (COCOMO, COCOMO II, SEER-SEM, etc.) vary in the parametric factors they employ. To best understand and test the details of a software estimate, the analyst will need to understand the specific assumptions and inputs required by the model used by the contractor. Specific information on COCOMO II may be found in the 5/03 Handbook for Software Cost Estimation published by the Jet Propulsion Laboratory.



## Software Cost Estimation Terminology

**Actual Cost of Work Performed (ACWP):** The actual direct costs incurred on a project at any given time.

**Algorithmic Models (also known as parametric models):** produce a cost estimate using one or more mathematical algorithms using a number of variables considered to be the major cost drivers. These models estimate effort or cost based primarily on the hardware/software size, and other productivity factors known as cost driver attributes.

**Analogy:** A method of estimating developed on the basis of similarities between two or more programs, systems, items etc.

**Analogy (or Comparative) Models:** Models that use a method of estimating that compares a proposed project with one or more similar and completed projects where costs and schedules are known. Then, extrapolating from the actual costs of completed projects, the model(s) estimates the cost of a proposed project.

**Annual Change Traffic (ACT):** The fraction of a software product's source instructions which undergoes change during a year, either through addition or modification. The ACT is the quantity used to determine the product size for software maintenance effort estimation.

**Baseline:** An established, fixed version of the project plan against which actual implementation of the project is measured.

**Bottoms-Up (Engineering Estimate or Grass Roots) Models:** A method of estimation that estimates each component of the project separately, and the results are combined ("rolled up") to produce an estimate of the entire project.

**Budgeted Cost of Work Performed (BCWP) (Earned Value):** The total value of work performed at any given time.

**Budgeted Cost of Work Scheduled (BCWS) (Budgeted Cost To-Date):** The total budgeted cost for work scheduled to be completed at any given time.

**Calibration:** A technique used to allow application of a general cost model to a specific set of data. This is accomplished by calculating adjustment factor(s) to compensate for differences between the referenced historical costs and the costs predicted by the cost model using default values.

**Computer-Aided Software Engineering (CASE):** Identifies a sector of the computer software industry concerned with producing software development environments and tools. The main components of a CASE product are

individual tools that aid the software developer or project manager during one or more phases of software development (or maintenance). Other features are a common user interface; interoperability of tools; and a repository or encyclopedia to provide a common tool base and central project database. CASE may also provide for code generation.

**Computer Software Component (CSC):** Since CSCIs may contain over 100,000 lines of code, they are further partitioned into computer software components (CSCs) and computer software units (CSUs).

**Computer Software Configuration Item (CSCI):** An aggregation of computer software that satisfies an end-use function and is designated for configuration management. A CSCI may be broken down into CSCs and/or CSUs.

**Computer Software Unit (CSU):** The lowest level in a breakdown of a software product.

**Constructive COSt MOdel (COCOMO):** A software cost estimation model developed by Barry Boehm and is described in his book, *Software Engineering Economics*.

**Cost Analysis:** The review and evaluation of the separate cost elements and proposed profit of (a) an offeror's or contractor's cost or pricing data and (b) the judgmental factors applied in projecting from the data to the estimated costs in order to form an opinion on the degree to which the proposed costs represent what the cost of the contract should be, assuming reasonable economy and efficiency.

**Cost Driver Attributes:** Productivity factors in the software product development process that include software product attributes, computer attributes, personnel attributes, and project attributes.

**Cost Drivers:** The controllable system design or planning characteristics that have a predominant effect on the system's costs. Those few items, using Pareto's law, that have the most significant cost impact.

**Cost Estimating Relationship:** An algorithm relating the cost of an element to physical or functional characteristics of that cost element or a separate cost element; or relating the cost of one cost element to the cost of another element.

**Cost Estimating Relationships (CER):** A mathematical expression that describes, for predicative purposes, the cost of an item or activity as a function of one or more independent variables.

**Cost Model:** An estimating tool consisting of one or more cost estimating relationships, estimating methodologies, or estimating techniques used to predict the cost of a system or one of its lower level elements.

**Cost or Pricing Data:** All facts that, at the time of the price agreement, the seller and buyer would reasonably expect to affect price negotiations. Cost or pricing data requires certification. Cost or pricing data are factual, not judgmental data, and are therefore verifiable. While these data do not indicate the accuracy of the prospective contractor's judgment about estimated future costs or projections, they do include the data utilized to form the basis for that judgment. Cost or pricing data are more than historical accounting data; they are all the facts that can be reasonably expected to contribute to the soundness of estimates of future costs and to the validity of determinations of costs already incurred.

**Cost/Schedule Control System Criteria (C/SCSC):** A set of criteria specified by the Federal Government for reporting project schedule and financial information since 1960. However, the American National Standards Institute/Electronic Industry Association as a new standard, called ANSI/EIA 748 or Earned Value Management Systems (EVMS).

**Delivered Source Instructions (DSI):** The number of source lines of code developed by the project. The number of DSIs is the primary input to many software cost estimating tools. The term delivered is generally meant to exclude non-delivered support software such as test drivers. The term source instructions includes all program instructions created by project personnel and processed into machine code by some combination of preprocessors, compilers, and assemblers. It excludes comments and unmodified utility software. It includes job control language, format statements, and data declarations.

**Delphi Technique:** A group forecasting technique, generally used for future events such as technological developments, that uses estimates from experts and feedback summaries of these estimates for additional estimates by these experts until a reasonable consensus occurs. It has been used in various software cost-estimating activities, including estimation of factors influencing software costs.

**Detail Estimating: Grass Roots, Bottoms-Up:** The logical buildup of estimated hours and material by use of blue-prints, production planning tickets, or other data whereby each operation is assigned a time value.

**Domain:** A specific phase or area of the software life cycle in which a developer works. Domains define developers and users areas of responsibility and the scope of possible relationships between products. The work can be organized by domains such as Software Engineering Environments, Documentation, Project Management etc.

**Evolutionary Acquisition (EA) Model:** The EA Model is pretty similar in content to the Waterfall Model except it encourages prototyping. The underlying factor in EA is to field a well-defined core capability quickly in response to a validated requirement, while using a phased upgrade program to eventually enhance the system to provide the full system capability. This process is also referred to as evolutionary prototyping.

**Expert Judgment Models:** Use a method of software estimation that is based on consultation with one or more experts that have experience with similar projects. An expert-consensus mechanism such as the Delphi technique may be used to produce the estimate.

**Fair Price (See Also Reasonable Price):** From the perspective of a buyer, a fair price is a price that is in line with (or below) the fair market value of the contract deliverable (to the extent that fair market value can be approximated through price analysis). "Fair market value" is the price you should expect to pay, given the prices of bona fide sales between informed buyers and informed sellers under like market conditions in competitive markets for deliverables of like type, quality, and quantity. When data on probable performance costs are available, a separate test of "fairness" is whether the proposed price is in line with (or below) the total allowable cost of providing the contract deliverable. This cost would be the cost incurred by a well managed, responsible firm using reasonably efficient and economical methods of performance, plus a reasonable profit. From the perspective of a seller, a fair price is a price that is realistic in terms of the seller's ability to satisfy the terms and conditions of the contract.

**Fourth Generation Language (4GL):** Fourth generation languages are programming languages closer to human languages than typical high level programming languages. Most 4GLs are used to access databases. For example a typical 4GL command is "FIND ALL RECORDS WHERE NAME IS "SMITH"."

**Function Points:** Function Points are those pieces of code that perform some specific activity related to inputs, inquiries, outputs, master files, and external system interfaces.

**Historical Data:** A term used to describe a set of data reflecting actual cost or past experience of a product or process.

**Incremental Development:** The incremental development approach is a top down implementation of distinct functional elements of the product. The development of each increment is accomplished as a separate waterfall type of development. The incremental development methodology differs from



the evolutionary approach in that under the incremental strategy the end product is well-defined.

**Knowledge Base:** The repository of knowledge in a computer system or organization. The collection of data, rules, and processes that are used to control a system, especially one using artificial intelligence or expert system methods.

**Life Cycle:** The stages and process through which hardware or software passes during its development and operational use. The useful life of a system. Its length depends on the nature and volatility of the business, as well as the software development tools used to generate the databases and applications.

**Management Information Systems:** A computer-based system of processing and organizing information that provides different levels of management within an organization with accurate and timely information needed for supervising activities, tracking progress, making decisions, and isolating and solving problems.

**Metric:** Quantitative analysis values calculated according to a precise definition and used to establish comparative aspects of development progress, quality assessment or choice of options.

**New Line of Code:** A source line of code that will be developed completely, i.e., designed, coded and tested.

**Paradigm:** A model, example, or pattern. A generally accepted way of thinking.

**Parametric Cost Model:** A mathematical representation of parametric cost estimating relationships that provides a logical and predictable correlation between the physical or functional characteristics of a system, and the resultant cost of the system. A parametric cost model is an estimating system comprising cost estimating relationships (CERs) and other parametric estimating functions, e.g., cost quantity relationships, inflation factors, staff skills, schedules etc. Parametric cost models yield product or service costs at designated levels and may provide departmentalized breakdown of generic cost elements. A parametric cost model provides a logical and repeatable relationship between input variables and resultant costs.

**Platform:** Hardware or software architecture of a particular model or family of computers. The term sometimes refers to the hardware and its operating system.

**Price Analysis:** The process of examining and evaluating a proposed price without evaluating its separate cost elements and proposed profit.

**Procedures:** Manual procedures are human tasks. Machine procedures are lists of routines or programs to be executed, such as described by the Job Control Language (JCL) in a mini or mainframe, or the batch processing language in a personal computer.

**Process:** The sequence of activities (in software development) described in terms of the user roles, user tasks, rules, events, work products, resource use, and the relationships between them. It may include the specific design methodology, language, documentation standards etc.

**Rapid Prototyping:** The creation of a working model of a software module to demonstrate the feasibility of the function. The prototype is later refined for inclusion in a final product.

**Rayleigh Distribution:** A curve that yields a good approximation to the actual labor curves on software projects.

**Real-Time:** 1) Immediate response. The term may refer to fast transaction processing systems in business; however, it is normally used to refer to process control applications. For example, in avionics and space flight, real-time computers must respond instantly to signals sent to them. 2) Any electronic operation that is performed in the same time frame as its real-world counterpart. For example, it takes a fast computer to simulate complex, solid models moving on screen at the same rate they move in the real world. Real-time video transmission produces a live broadcast.

**Reasonable Price (See Also Fair Price):** A price that a prudent and competent buyer would be willing to pay for the contract deliverable, given adequate data on (1) market conditions, (2) alternatives for meeting the requirement, (3) the evaluated price of each alternative, and (4) non-price evaluation factors (in "best value" competitions).

**Re-engineering:** Process of restructuring and redesigning an operational (or coded) hardware or software system or process in order to make it meet certain style, structure, or performance standards.

**Reusability:** Ability to use all or the greater part of the same programming code or system design in another application.

**Reuse:** Software development technique that allows the design and construction of reusable modules, objects, or units, that are stored in a library or database for future use in new applications. Reuse can be applied to any

methodology in the construction phase, but is most effective when object oriented design methodologies are used.

**Security:** The protection from accidental or malicious access, use, modification, destruction, or disclosure. There are two aspects to security, confidentiality and integrity.

**Software Development Life Cycle:** The stages and process through which software passes during its development. This includes requirements definition, analysis, design, coding, testing, and maintenance.

**Software Development Life Cycle Methodology:** Application of methods, rules, and postulates to the software development process to establish completeness criteria, assure an efficient process, and develop a high quality product.

**Software Engineering Institute (SEI):** SEI is a federally funded research and development center established in 1984 by the DoD with a broad charter to address the transition of software engineering technology. The SEI is an integral component of Carnegie Mellon University and is sponsored by the Office of the Under Secretary of Defense for Acquisition and Technology. SEI developed the Software Acquisition Capability Maturity Model (CMM) and the Checklist and Criteria for Evaluating the Cost and Schedule Estimating Capabilities of Software Organizations.

**Software Method (or Software Methodology):** Focuses on how to navigate through each phase of the software process model (determining data, control, or uses hierarchies; partitioning functions; and allocating requirements) and how to represent phase products (structure charts; stimulus-response threads; and state transition diagrams).

**Software Tool:** Program that aids in the development of other software programs. It may assist the programmer in the design, code, compile, link, edit, or debug phases.

**Source Lines of Code (SLOC):** All executable source code statements including deliverable Job Control Language (JCL) Statements, Data declarations, Data Typing statements, Equivalence statements, and Input/Output format statements. SLOC does not include any statement that upon its removal, the program will still compile, e.g., comments, blank lines, and non-delivered programmer debug statements.

**Space Systems Cost Analysis Group:** The SSCAG is an organization co-chaired by the cost directorates of the Air Force Space and Missile Systems Center (SMC) and NASA at Johnson Space Center. The Software Subgroup

of the SSCAG, under a multi-year task, developed the Software Methodology Handbook and a software database of over 2600 records used to calibrate five software estimating models (PRICE-S, SEER-SEM, SLIM, SASET AND REVIC).

**Spiral Development:** The spiral model encompasses features of the phased life cycle as well as the prototype life cycle. However, unlike those life cycles, the spiral model uses risk analysis as one of its elements. It also uses the waterfall model for each step so as to avoid any risks.

**Top-Down Models:** Use a method of estimation that estimates the overall cost and effort of the proposed project derived from global properties of the project. The total cost and schedule is partitioned into components for planning purposes.

**Update:** To update an estimate or CER means to utilize the most recent data to make it current, accurate and complete.

**Validation:** In terms of a cost model, a process used to determine whether the model selected for a particular estimate is a reliable predictor of costs for the type of system being estimated.

**Waterfall Model:** An eight-phase process used in developing software for most Department of Defense (DoD) weapon systems, as described in DoD-Standard 2167A. This process, when done sequentially, is based on the waterfall model of software development, as described by Barry Boehm. Each phase requires the delivery of particular documentation products.

**Work Breakdown Structure:** A work breakdown structure is a product-oriented family tree, composed of hardware, software, services, data and facilities which results from system engineering efforts during the development and production of a defense material item, and which completely defines the program. A work breakdown structure displays and defines the product(s) to be developed or produced and relates the elements of work to be accomplished to each other and to the end product. MIL-HDBK 881 is the modern guide for developing a WBS. (See Appendix 19C contains a sample WBS.)

**Workstation:** High-performance, single user microcomputer or minicomputer that has been specialized for graphics, CAD, CAE, or scientific application.



## Popular Parametric Software Cost Models

### 19B.1 INTRODUCTION

There are several sophisticated parametric software cost models that consider multiple parameters in computing cost or effort required. A list of some common models is provided below, listed in alphabetical order. Subsequent sections are a basic discussion of each model. For each of the models, general information, principal inputs, processing, principal outputs, calibration, and life cycle (or support) considerations will be discussed. When more detailed information is needed, cost estimators are encouraged to consult the referenced documents.

- Automated Cost Estimating Integrated Tools (ACEIT)
- COConstructive COSt MOdel (COCOMO.II)
- Cost Xpert
- ForeSight
- KnowledgePLAN Model
- PRICE-S Model
- Revised Enhanced Version of Intermediate COCOMO (REVIC)
- SoftCost-Object-Oriented (OO) Model
- Software Architecture, Sizing, and Estimating Tool (SASET)
- Software ESTimator (SoftEST)
- Software Life Cycle Model (SLIM)
- System Evaluation and Estimation of Resources Software Estimating Model (SEER-SEM)

### 19B.2 AUTOMATED COST ESTIMATING INTEGRATED TOOLS (ACEIT)

ACEIT is an estimating system consisting of a suite of tools designed to assist cost analysts in arriving at cost estimates, conducting “what-if?” studies, developing cost proposals and evaluations, conducting risk and uncertainty analysis, and developing **Cost Estimating Relationships (CERs)**. Its primary purpose is Financial Management. ACEIT is a Joint Service system, sponsored by the **Air Force Materiel Command (AFMC)** Electronic Systems Center and the US Army Cost and Economic Analysis Center (**CEAC**). The result of government-sponsored efforts, the ACEIT suite of applications is available to

U.S. Government organizations with no charge for use (but there is an annual maintenance & support fee). It is used by Acquisition Program Offices and at various other levels across the Armed Services, throughout DoD, and by other Government agencies.

The ACEIT system is comprised of five analysis/estimating tools:

- Automated Cost Estimator (ACE), (the spreadsheet)
- Automated Cost Database (ACDB), (library of commercial and non-commercial cost models)
- Cost Analysis Statistics Package (CO\$TAT), (full-feature statistics package)
- Cost Risk Assessment (RI\$K), (model which quantifies risk associated with a cost estimate)
- ACE Executive.

ACE is the heart of ACEIT. It automates all of the steps of the estimating process, including, building a **Work Breakdown Structure (WBS)**, specifying estimating methods, performing learning, time phasing and inflation, and documentation. ACE also provides access to on-line databases and a knowledge base of over 1000 cost estimating relationships, models, and source documents from which users can identify appropriate estimating methods and incorporate them into their estimate.

**The ACEIT system** is comprised of five analysis/estimating tools: Automated Cost Estimator (ACE), (the spreadsheet); Automated Cost Database (ACDB), (relational data base); Cost Analysis Statistics Package (CO\$TAT), (full-feature statistics package); Cost Risk Assessment (RI\$K), (quantifies risk associated with a cost estimate) and ACE Executive. ACE, the Automated Cost Estimator, is a special purpose program, specifically developed for cost analysis. It has been organized and structured to follow the steps or phases used in developing a cost estimate, ranging from defining what is being estimated through time-phasing and final documentation. ACDB is the cost database module that contains cost, schedule, technical, and programmatic data on system acquisition contracts. It is a PC-based program designed to assist the user in building, loading, maintaining, and querying a relational data base of cost and technical data, and in analyzing data subsets retrieved from it. The ACEIT Executive is a three-part application: the ACE Calculation Server, the RI\$K Calculation Server, and the Microsoft Excel Client. The Server applications let you run ACE and RI\$K sessions from any client application. Using the Excel client, you can link applications that would naturally interface with Excel to ACE and RI\$K. The user can either enter own CERs/equations or import the results of other models such as PRICE and SEER. Outputs look like spreadsheets or can link to Excel.

ACDB is an automated cost database with the capability to enter, search, and retrieve cost, schedule, technical, and programmatic data and to automatically load retrieved data into the **Cost Analysis Statistics Package (CO\$TAT)**. CO\$TAT is a cost analysis statistical package built specifically for the cost estimator to perform statistical analyses commonly used in cost estimation. RISK is a cost risk application that performs risk and uncertainty analysis. ACE Executive allows estimates and models hosted in ACE to act as cost model servers to other applications.

### 19B2.1 ACEIT Inputs

ACEs provides a workscreen or spreadsheet to perform each step required to make an estimate. The complete list of steps is as follows:

1. Define the WBS/CES
2. Enter the estimating methodologies
3. Adjust G&A, fee, overhead, escalation, units
4. Apply learning curves
5. Time phase the estimate
6. Estimate/perform what-ifs
7. Document the estimate
8. Review and Refine

The input screens resemble spreadsheets and require the equations and variables to be defined. Within each workscreen, functionality is provided in ACE by giving each column a dedicated purpose. For example, the Learning Curve Slope column only lets you enter a learning curve slope or a variable name that represents the slope. Each column has its own syntax. By making entries in several columns for a row, you can specify that rows estimating methodology. Typically, the rows in ACE represent WBS/CES line items or input variable items.

ACE can import from PRICE Software Models, SEER Cost Models, MS Excel and MS Word.

### 19B2.2 ACEIT Processing

Since the user enters the equations and variables, the user basically defines the processing. The ACE structure allows the user to progressively move from one workscreen to another. The user creates the model structure of the estimate, and calculates the result. This model building can be free-form. The user does not need to move through the workscreens in a pre-defined order. Unique IDs or variable names can be specified for each row and then

used in the ACE columns of other rows. This lets rows of the model link together into a system of equations.

### 19B2.3 ACEIT Outputs

Reports are basically spreadsheets containing the results. Tools that ACE can export data to include MS Excel and MS Word.

### 19B2.4 ACEIT Calibration

Since the user controls the equations, calibration is accomplished by creating historical databases, experience and refining the equations.

### 19B2.5 ACEIT Life Cycle Considerations

Include the WBS Lifecycle elements into the model.

## 19B.3 CONSTRUCTIVE COST MODEL (COCOMO.II)

COCOMO.II is a screen-oriented, interactive-software package that assists in budgetary planning and schedule estimation of a software development project prior to any work beginning. Through the flexibility of COCOMO.II, a software project manager can develop a model (or multiple models) of projects in order to identify potential problems in resources, personnel, budgets, and schedules both before and after the potential software package has been completed.

The COCOMO.II software package is based upon a recently revised version of the original **Constructive Cost Model (COCOMO)** first published by Barry Boehm in his book *Software Engineering Economics*, Prentice-Hall (1981), and the Ada COCOMO (1987) predecessors.

**The primary objectives of the COCOMO.II.1997 effort were to:**

- To develop a software cost and schedule estimation model attuned to the life cycle practices of the 1990's and 2000's.
- To develop software cost database and tool support capabilities for continuous model improvement.
- To provide a quantitative analytic framework, and set of tools and techniques for evaluating the effects of software technology improvements on software life cycle costs and schedules.

The full COCOMO.II model includes three stages. Stage 1 supports estimation of prototyping or applications composition efforts. Stage 2 supports estimation in the Early Design stage of a project, when less is known about the project's cost drivers. Stage 3 supports estimation in the Post-Architecture stage of a project. The current version of COCOMO.II implements Stage 3 formulas



to estimate the effort, schedule, and cost required to develop a software product. It also provides the breakdown of effort and schedule into software life cycle phases and activities from the original COCOMO manual. These are still reasonably valid for waterfall model software projects, but need to be interpreted for non-waterfall projects.

Equation 19B-1 below is the equation used by COCOMO.II to calculate the estimated effort.

**Equation 19B-1. Effort Estimation Equation**

$$PM = \prod_{i=1}^{17} (EM_i) \cdot A \cdot \left[ \left( 1 + \frac{BRAK}{100} \right) \cdot Size \right]^{1.01 + 0.01 \sum_{j=1}^5 SF_j} + \left( \frac{ASLOC \cdot \left( \frac{AT}{100} \right)}{ATPROD} \right)$$

Where:

$$Size = KNSLOC + \left[ KASLOC \cdot \left( \frac{100 - AT}{100} \right) \cdot \frac{AA + SU + 0.4 \cdot DM + 0.3 \cdot CM + 0.3 \cdot IM}{100} \right]^{B = 1.01 + 0.01 \sum_{j=1}^5 SF_j}$$

Symbol	Description
A	Constant, provisionally set to 2.5
AA	Assessment and assimilation
ADAPT	Percentage of components adapted (represents the effort required in under-standing software)
AT	Percentage of components that are automatically translated
ATPROD	Automatic translation productivity
BRAK	Breakage: Percentage of code thrown away due to requirements volatility
CM	Percentage of code modified
DM	Percentage of design modified
EM	Effort multipliers: RELY, DATA, CPLX, RUSE, DOCU, TIME, STOR, PVOL, ACAP, PCAP, PCON, AEXP, PEXP, LTEX, TOOL, SITE
IM	Percentage of integration and test modified
KASLOC	Size of the adapted component expressed in thousands of adapted source lines of code
KNSLOC	Size of component expressed in thousands of new source lines of code
PM	Person Months of estimated effort
SF	Scale Factors: PREC, FLEX, RESL, TEAM, PMAT
SU	Software understanding (zero if DM = 0 and CM = 0)

**Product Description:**

COCOMO.II model is an updated version of the 1981 COCOMO and 1987 Ada COCOMO models tailored to the new software development life cycle processes and capabilities. Major new modeling capabilities of COCOMO.II include: a tailorable family of software sizing models, involving Object Points, Function Points, and Source Lines of Code (SLOC); non-linear models for software reuse and reengineering; an exponent-driver approach for modeling relative software diseconomies of scale; and several changes to previous COCOMO effort-multiplier cost drivers.

**19B.3.1 COCOMO.II Inputs**

The primary COCOMO.II input are the program size, in KDSI, Function Points or Object Points. However, ratings for sixteen additional attributes must be assessed. These attributes are included in four categories as follows:

- **Product attributes:** These attributes describe the environment in which the program operates. The attributes in this category are: reliability requirements, database size, documentation matched to life cycle needs, required reusability and program complexity.
- **Platform attributes:** These attributes refer to the limitations placed upon the development effort by the hardware and operating system being used to run the project. The attributes in this category are execution time constraints, main storage constraints, and platform volatility.
- **Personnel attributes:** These attributes describe the skill levels of personnel assigned to the program. The attributes in this category include: analyst capability, applications experience, programmer capability, programming language experience, platform experience and personnel continuity.
- **Project attributes:** These attributes refer to the constraints and conditions under which project development takes place. The attributes in this category are use of software development tools and multi-site development.

These 16 factors (or **effort multipliers (EM)**) are incorporated into the schedule and effort estimation formulas by multiplying them together. The numerical value of the *i*th adjustment factor is called  $EM_i$  and their product is called the adjustment factor or EAF. The actual effort,  $PM_{total}$ , is the product of the nominal effort times the EAF.

**19B.3.2 COCOMO.II Processing**

Using COCOMO.II, a nominal assessment of man-months based on size alone is assessed for the program being considered. Next, the ratings for all

attributes are multiplied to compute the required man-months of effort for the project. The primary challenges in using COCOMO.II are determining size and assigning proper ratings to the sixteen attributes.

### **19B.3.3 COCOMO.II Outputs**

The output of the COCOMO.II model is simply the level of effort in man-months for the project being estimated and a schedule in months. The effort output can easily be converted to a monetary value if the cost per man-month is known. The Phase Distribution is one of the outputs. Its function is to display a breakdown of the software effort and schedule into the phases of the development cycle. These phases are plans & requirements, design, programming and integration & test. The outputs of the model are very basic and not very flexible, so performance metrics will need to be created outside of this model.

### **19B.3.4 COCOMO.II Calibration**

Calibration is essential to the proper use of software cost models. The user of the COCOMO.II may calibrate EAFs and the effort/schedule equations of the current project. (Detailed procedures for COCOMO calibration are discussed in the COCOMO Reference Manual.)

### **19B.3.5 COCOMO.II Life Cycle Considerations**

There are no life cycle considerations included in the COCOMO.II model. The COCOMO Maintenance model (COCOMO-M), described in Chapter 30 of Boehm's book, can be used to estimate annual man-months required to support a software program. No mention of COCOMO-M is contained in the COCOMO.II literature. For the intermediate level COCOMO-M, all inputs from COCOMO 1.0 are used, except that different numerical values are assigned to two attributes: reliability and use of modern design practices. The annual support costs can be computed by multiplying the number of man-months by the average cost of a man-month.

## **19B.4 COST XPERT**

Marotz, Inc. developed a new model called Cost Xpert. The model is designed to provide software project cost estimates, to determine optimal delivery time, to break the project down into tasks by allocating time/effort and to perform quantitative risk/sensitivity analysis. It supports projects using object-oriented development, GUI development and formalized software reuse.

### **19B.4.1 Cost Xpert Inputs**

The most important input parameter is size, which can be input as SLOC or function points, GUI Metrics, top down and bottoms up methodologies.

**Product Description:**

Cost Xpert is an automated tool that supports software costing, scheduling and risk assessment using SLOC, function points, GUI Metrics, top down and bottoms up methodologies. Cost Xpert consists of five main notebook tabs: Project, Volume, Environment, Constraints and Results. Outputs include an input data summary, software development cycle task report, risk report, labor report by labor category, maintenance report, and documentation deliverables report.

All size inputs are converted to equivalent SLOC. When more than one size estimating methodology is used, the user can select which results to be averaged together to create the cost and schedule estimate. Cost Xpert has about thirty inputs which are included in the three other “notebooks tabs”. A help screen for each input assists the user in selecting proper values. Each of the tabs is now summarized.

- **Project Tab:** Inputs in this tab include primary and secondary programming language; project coefficients (commercial, military, embedded etc.); project standards (commercial, DoD-STD-2167A etc.) identifies documentation required; project type (commercial, embedded etc.) used to identify likely risk factors/ defect rates; and project life cycle (client-server, standard-small etc.) used to relate activities to effort. These inputs can all be tailored.
- **Environment Tab:** The user inputs the factors that influence the efficiency of the software development team such as analyst and programmer capability; applications, virtual machine and language experience; execution time and main storage constraints; virtual machine and requirements volatility; computer turn around time; database size; product complexity and reliability; required reuse and security; and use of modern programming practices and tools. Each of these can be in five levels rated from very low to very high with the definitions and numerical values included on the tab view.
- **Constraints Tab:** The user can use the constraints tab to assign numerical values to eight constraint areas relevant to the project. These areas are time-cost trade-off, review time, requirements analysis, minimum review time, Beta testing, cushion, overlap and risk tolerance. The estimated percentages can be assigned by the user or left as a default value.



### 19B.4.2 Cost Xpert Processing

The model performs all computations based on “equivalent SLOC”. The core costing equations involve a relatively simple equation of the form “Effort =  $\alpha$  \* volume  $\beta$ .”  $\alpha$  and  $\beta$  are organization specific coefficients and can be adjusted by the user during calibration. The exact equations used in the model are not defined in the *User’s Guide*. The environmental factors used in Cost Xpert are very similar to those used in the REVIC model.

### 19B.4.3 Cost Xpert Outputs

The model’s primary outputs are development effort and schedule, including the optimal effort and schedule for the program(s) being analyzed. Cost Xpert provides staffing profiles by labor category/month. Cost Xpert also has a risk assessment with which the user can perform numerous size, effort, and schedule risk analyses, as well as identifying likely risks to the project and a sensitivity analysis. Maintenance and document deliverable reports summarize those areas. The schedule report is exportable to MS Project. There is a risk assessment tool available (Risk Xpert) that supports both risk assessment and contingency planning/tracking to mitigate risks in a formal, optimized manner.

### 19B.4.4 Cost Xpert Calibration

Cost Xpert contains a minimum amount of detail for calibrating the model. Basically, it is done by comparing actual historical data to model predicted results, then adjusting the coefficients  $\alpha$  and  $\beta$ . Further calibration is made by tailoring the inputs in the Project and Constraints tabs.

### 19B.4.5 Cost Xpert Life Cycle Considerations

The model predicts both the maintenance effort for each year of the project and the projected maintenance adjusted for inflation. The quantity of software defect estimates and support calls are also estimated based on the original project input data.

## 19B.5 FORESIGHT MODEL

ForeSight is a parametric tool for forecasting time, effort, and cost of non-military software projects; the projects can be new developments, modifications, integration of off-the-shelf applications, maintenance programs, or software upgrades of any type. ForeSight is a tool for all project management tasks related to effort and time forecasting--budgeting, staffing plans, estimate to complete, performance measurement etc. This model has a one button OLE interface to MS Project.

ForeSight is a contemporary adaptation of the PRICE S model to address non-military software projects only. It was based upon the same database

used to construct PRICE S. ForeSight, however, was tuned to the commercial and non-military government (AIS) records of the PRICE S database and has some different equations. In addition, several databases unique to ForeSight development were incorporated, among them: the **National Software Data Information Repository (NSDIR)**, the **ISBSG (International Software Benchmarking Standards Group)** database, and a proprietary database of **Object Oriented (OO)** applications.

**Product Description:**

ForeSight is a software tool for forecasting time, effort, and cost of non-military software projects; the projects can be new developments, modifications, integration of off-the-shelf applications, maintenance programs, or software upgrades. Project sizing metrics include: SLOC, function points and predictive object points (POPs). ForeSight produces estimates that include: size, schedules, staffing, labor effort, cost, and quality. This model has a one-button interface to MS Project for file creation, importing and updating. It tracks milestones and benchmarks the project against past history. ForeSight supports waterfall, spiral, evolutionary and incremental development projects. It allows Risk Analysis and will generate an Estimate To Complete (ETC).

### 19B.5.1 ForeSight Inputs

One of the primary inputs for the ForeSight model is program size. Size is usually expressed in SLOC that can be input directly by the user or computed using a sizing model included in the ForeSight package. The Sizer model estimates size by SLOC, by using **predictive object points (POPs)** or by converting a function point count to SLOC. Other key inputs include the following:

- **Application:** The seven basic functional categories of the inherent software instruction difficulty are: mathematical, string manipulation, data storage and retrieval, on-line, real-time, interactive, and operating system. The result of this classification procedure is summarized in a single value called Application.
- **Productivity factor:** A parameter that can be calibrated which relates the software program to the organizational capabilities, experience, and individual talents of the team that will perform the software development. This factor should be based upon historical data.
- **Complexity:** Complexity describes the effects of additional factors affecting the development environment that are directly related to schedule or time, such as a development tools, personnel skills, and potential requirements growth.

- **Platform:** Platform relates the cost of software development to the requirements of the environment in which the software must operate. It is a measure of the transportability, reliability, testing, and documentation required by the contract.
- **Utilization:** The proportion of the processor capability used, relative to its available speed and memory capacity, is represented by the variable called Utilization.
- **Level of New Design and Code:** Percentage of the end product that will require new design and/or coding effort. This input considers the degree of modification or reuse for the software.

There are other inputs for internal and external integration effort, schedule start and end dates, programming language used, and economic factors.

### 19B.5.2 ForeSight Processing

ForeSight uses a common core equation that relates schedule and effort to software product size and software production capability. The form of the core equation is:

$$M = A * \text{Size}^B;$$

where  $M$  is either effort or time and  $A$  and  $B$  are functions of software production capability and/or product type. ForeSight adjusts the size measure (in units of lines of code, function points, or object points) for software functionality and amount of reuse. Other areas around the core equation that influence project performance are treated by the forecasting engine. The exact equations are proprietary and are not included in the user's manual.

### 19B.5.2 ForeSight Outputs

Results are provided in a number of fix formatted forms, including text and graphic reports, screens, Microsoft Project inputs, and a ForeSight project file in Microsoft Access format. ForeSight produces estimates that include: size, schedules, staffing, labor effort, cost, and quality.

### 19B.5.3 ForeSight Calibration

The HELP menu provides no indication of how to calibrate this model other than by changing the individual attribute ratings or through updating "historical elements". Historical elements are software systems, subsystems or elements that have been completed and for which cost, schedule or effort information is available. A historical element may be integrated into a new system/subsystem as COTS or as an analogy for forecasting purposes. "In either case ForeSight will calibrate the technical and cost information to establish a unique performance measure for each historical element."

### 19B.5.4 ForeSight Life Cycle Consideration

The ForeSight Life Cycle Model, included in the ForeSight package, is a detailed model which computes software operation and support costs. The Profile menu generates a graphical display that is used to control the distribution of effort expended during the support period. The total effort is a combination of four distributions: Defect Detection, Defect Repair, Enhancement, and Adaptation. All four may be viewed on a single Profile screen at once. Associated with every element of an ForeSight Estimation Breakdown Structure is a table of data that enables tailoring of the type listed above. Supplemental data controls the level of tasks and resources **estimated** for elements.

### 19B.6 KNOWLEDGEPLAN MODEL (REPLACES CHECKPOINT)

The KnowledgePLAN model algorithms are derived from over 6,700 projects. The model is applicable to all types of programs and has its knowledge base updated annually.

#### Product Description:


KnowledgePLAN is a tool that guides the user through the development of a software project estimate and plan using a large knowledge base that is updated annually. KnowledgePLAN combines knowledge-based estimation, "what if" analysis and scheduling functionality within one tool. Work Breakdown Structures can be integrated into a Project (MPX compatible) management tool and allows the PM tool to vary schedules, task assignments and resource allocations. Allows project sizing by SLOC, Function Point and Analogy. KnowledgePLAN creates and refines detailed project plans for a bi-directional interface with Microsoft Project or other enterprise project management system. The tool will track milestones, schedules, resources, actual work effort and defects found.

#### 19B.6.1 KnowledgePLAN Inputs

Like the other models discussed, the primary input for KnowledgePLAN is size. However, KnowledgePLAN is different than most other models in that it works primarily in sizing by analogy or with function points instead of SLOC. The model will accept SLOC, but converts SLOC to function points using conversion factors in the model. In addition to size, the model requires certain inputs for a quick (or basic) estimate, and additional parameters for a detailed estimate. The inputs for each option are now summarized.

- **Quick Estimate Inputs:** The following inputs are required for both quick and detailed estimates:
  - ⇒ **Project description information.**
  - ⇒ **Project nature:** Whether it is a new program, an enhancement, a conversion, reengineering, maintenance, etc.



- 
- ⇒ **Project scope:** Whether the program is a stand-alone program, program within a system, a disposable prototype, etc.
  - ⇒ **Project Topology:** Whether the project is being implemented as a stand-alone platform, local or wide area network, a client server, or a distributed network.
  - ⇒ **Project class:** Who is the customer/end-user of the software project such as contract, commercial, government, IT/MIS and is it a single site, multi-site or network.
  - ⇒ **Project type:** What kind of software is the deliverable; non-procedural application or system with subtypes such as off-line processing, interactive graphical user interface (GUI), etc.
  - ⇒ **Software Products:** For sizing by analogy, which system/application is similar in size (small, medium, large) with similar applications such as business, generic, billing, etc.
  - **Detailed Estimate Inputs:** KnowledgePLAN has over 100 factors, or project attributes, that can be used to fine-tune the model's estimates. These are categorized into personnel, technology, process, environmental, product and maintenance which are briefly described. Each factor is rated essentially from "very low" to "very high" in five possible steps with "3" being average or typical. Lower ratings generally result in increased effort and schedule. If the user has no information for a factor, he or she can leave it as "not answered". The model keeps track of how many attributes have been answered (i.e., 25%) and this is included in the tool's risk assessment.
    - ⇒ **Personnel attributes:** These attributes are divided into four subsets: project management, development experience, user personnel experience, and quality personnel experience. Examples of project management attributes are organization structure, team morale, and project management experience.
    - ⇒ **Technology attributes:** These attributes address the impact of Life Cycle software tools and hardware platforms on the development environment. Examples of attributes are design automation environment, project documentation library, development hardware stability, and terminal response time.
    - ⇒ **Process attributes:** These attributes address the life cycle methodology used on the project that affects development and

quality assurance. Examples are analysis, life cycle, quality, testing, and documentation.

⇒ **Environment attributes:** These attributes address the importance of organization and office factors.

⇒ **Product factors:** The Products Attributes dialog box allows the user to characterize any constraints that may affect the ability to fulfill user requirements such as security or performance. The Products Attributes dialog box has two tabs: Requirements and Architecture.

### 19B.6.2 KnowledgePLAN Processing

The specific algorithms used in this model are not defined in the KnowledgePLAN User's Guide. The Guide does state that the project estimates are regulated by: algorithms from the project's assigned knowledge base, by task category inclusion rules, by adjustment rules from the estimation engine, by task/category properties and by task estimation flags. The starting point is the knowledge base assigned to the project by the model based primarily on the answers to the quick estimate inputs. Knowledge bases contain the rules and estimates for the base estimate. The algorithms are tuned to the default settings of the project classification variables and can be viewed/modified by the user.

### 19B.6.3 KnowledgePLAN Output

The model provides the user with several different outputs. Like most other models, KnowledgePLAN provides an estimate of schedule, staffing, and effort in dollars or person-months in table or Gantt chart views. KnowledgePLAN provides a risk analysis of various input options selected; along with estimates for size, defects, test and reliability, maintenance, documentation, and various productivity parameters.

### 19B.6.4 KnowledgePLAN Calibration

The user can calibrate KnowledgePLAN by overriding the model's default values for inputs or quality and productivity rates. This can either be done directly or, more commonly, by creating templates based on historical data. Templates include any values for which the user wants to override the model's defaults, and are requested as part of a basic estimate. Templates pass along the domain and knowledge base that the new project will use. The *KnowledgePLAN User's Guide* provides information about template creation.

### 19B.6.5 KnowledgePLAN Life Cycle Considerations

KnowledgePLAN includes more than fifteen "maintenance" inputs in the "detailed estimate inputs". Examples are maintenance personnel experience,

amount of replacement and restructure planning, number of sites, and customer support. The user can also specify the amount of code to be added or deleted annually. The model estimates the annual cost of maintenance and enhancements for a time period specified by the user, up to 20 years.

### 19B.7 PRICE-S MODEL

Martin Marietta Price Systems (initially RCA Price, then GE Price) originally developed this model as one of a family of models for hardware and software cost estimation. Developed in 1977, PRICE-S was the first commercially available detailed parametric software cost model to be extensively marketed and used. The PRICE-S model is proprietary; not all equations are published, although the *PRICE-S Reference Manual* describes the basic parametric relationships. The model is applicable to all types of software projects. It considers all eight phases of the software development cycle, plus system concept and operational testing phases.

#### Product Description:

The PRICE Software Model Suite (PRICE-S) is a parametric cost and scheduling model that consists of three models to estimate costs and schedules for the development, operation and support of computer software. The PRICE-S Acquisition Model is an application of PRICE empirical modeling methods to the problem of forecasting software cost and schedule. The PRICE-S Sizing Model utilizes three analytical estimating tools to estimate the size of the software being developed. The first tool is used to estimate SLOC based on inputs for qualitative descriptors, quantitative descriptors and sizing factors. The second tool is a Function Point sizer that converts the function point count to SLOC. The third tool is an Object Metric, Predictive Object Points (POPs) for sizing object oriented development projects. The PRICE-S Life Cycle Cost Model is used to develop early costing of the maintenance and support phase for a software project. The Acquisition Model provides the development cost and design parameters to the Life Cycle Cost Model, with the user inputting the support activity and support period. Support cost estimates include Corrective, Enhancement and Growth categories. ForeSight is a new, less expensive software cost model that is compatible with PRICE-S except for military/mission critical software development projects.

#### 19B.7.1 PRICE-S Inputs

One of the primary inputs for the PRICE-S model is program size. Size is usually expressed in SLOC that can be input directly by the user or computed using a sizing model included in the PRICE-S package. The Sizer model estimates size by SLOC, by using **predictive object points (POPs)** or by converting a function point count to SLOC. Other key inputs include the following:

- **Application.** The seven basic functional categories of the inherent software instruction difficulty are: mathematical, string

manipulation, data storage and retrieval, on-line, real-time, interactive, and operating system. The result of this classification procedure is summarized in a single value called Application.

- **Productivity factor.** A calibratable parameter that relates the software program to the organizational capabilities, experience, and individual talents of the team that will perform the software development. This factor should be based upon historical data.
- **Complexity.** Complexity describes the effects of additional factors affecting the development environment that are directly related to schedule or time, such as a development tools, personnel skills, and potential requirements growth.
- **Platform.** Platform relates the cost of software development to the requirements of the environment in which the software must operate. It is a measure of the transportability, reliability, testing, and documentation required by the contract.
- **Utilization.** The proportion of the processor capability used, relative to its available speed and memory capacity, is represented by the variable called Utilization.
- **Level of New Design and Code.** Percentage of the end product that will require new design and/or coding effort. This input considers the degree of modification or reuse for the software.

There are other inputs for internal and external integration effort, schedule start and end dates, programming language used, and economic factors.

### 19B.7.2 PRICE-S Processing

Parametric relationships which combine management perception and historical results are used to relate new software projects to costs and schedules that are typical of the work to be accomplished. Although much material concerning the PRICE-S algorithms has been published, some details concerning the algorithms are proprietary and are not available to the user. It is known that PRICE-S computes a “weight” of software based on the product of instructions and application inputs, which are comparable to hardware volume and density respectively. The productivity factor and complexity inputs are very sensitive parameters that affect effort and schedule respectively. Platform is known to be an exponential input; hence, it can be very sensitive. Other input parameters are used to adjust the “weight” of software for a specific program. A 1988 paper published by PRICE Systems, entitled *The Central Equations of the PRICE Software Cost Model*, describes many



internal algorithms in detail, although algorithms may have been modified since that time.

### 19B.7.3 PRICE-S Outputs

PRICE-S computes an estimate in person-months which may be converted to cost in dollars or other currency units. The model estimates schedule by milestones, with a staffing profile. In addition to cost and schedule estimates, PRICE-S provides automatic sensitivity and schedule effect analyses, together with monthly cost and progress summaries to support budgeting, risk analysis, and project tracking.

### 19B.7.4 PRICE-S Calibration

Organizational performance history serves as input to a calibration mode that fits the model to user specific environments that characterize productivity within a line of business. The PRICE-S model can be run in the ECIRP (PRICE backwards) mode to calibrate selected parameters. The most common calibration is that of the productivity factor, which, according to the *PRICE-S Reference Manual*, tends to remain constant for a given organization. It is also possible to calibrate platform, application, and selected internal parameters.

### 19B.7.5 PRICE-S Life Cycle Consideration

The PRICE-S Life Cycle Model, included in the PRICE-S package, is a detailed model which computes software operation and support costs. The Life Cycle Model is designed to be used in conjunction with the Acquisition Model that provides the development costs and design parameters to the Life Cycle Model. The primary inputs are support descriptors such as number of installations, expected growth, and quality and enhancement levels; three calibratable support productivity factors; and separate size and expected growth. The Life Cycle Model outputs cost in three support categories: maintenance, enhancements, modifications and growth. It also outputs a predicted number of delivered defects in the program to be supported.

## 19B.8 SOFTCOST-OBJECT-ORIENTED (OO) MODEL

The SoftCost Object-Oriented (SoftCost-OO) model, developed and marketed by Resources Calculations, Inc. (RCI), evolved from the SoftCost-Ada model developed by Don Reifer which was, in turn, based on the published work of Dr. Robert Tausworthe of the Jet Propulsion Laboratory. In addition to Ada, SoftCost-OO has calibration files for the C++ language and generic object-oriented paradigms. The model is markedly different from the SoftCost-R model, which more closely paralleled COCOMO. The SoftCost-OO model has a database of over 240 Ada, 50 C++, and 120 object-oriented projects. The model is one of a family of models marketed by RCI, which includes SoftCost-R (a software cost estimation tool for data processing, scientific and

real-time applications), Asset-R (a function point tool to estimate software size) and SSM (an analogy software sizing model).

SoftCost-OO includes a **work breakdown structure (WBS)** file editor which allows use of preloaded WBS files, modification of pre-loaded files, or development of a new WBS from scratch. The WBS file editor also checks for correct allocation of duration and effort percentages for each WBS item, and for an allowable set of predecessor activities in a network.

SoftCost-OO is applicable to all types of object-oriented programs and considers all phases of the software development cycle. The model's equations are published in the *SoftCost-OO User's Guide*; however, the computer program used to solve these equations and related analyses is proprietary to RCI.

**Product Description:**

SoftCost-OO is an object oriented and reuse software estimation tool. This tool is a PC-based parametric cost model consisting of a screen editor, estimation model, and outputs. Phases covered include software requirements through software testing, including hardware and software integration testing. There are three databases, Ada, C++, and Object-Oriented. SoftCost-OO contains five submodels, sizing, estimating, risk, allocation (effort/labor categories), and life cycle (maintenance). The sizing submodel calculates equivalent source lines of code for projects developed in Ada, C++, or a mix of languages. Function points can also be used. The estimating submodel develops an effort and schedule estimate for a project using parametric and size information provided by the user. The risk submodel allows the user to play "what-if" gaming scenarios by varying effort, schedule, and size. The allocation submodel takes the effort and schedule from the risk submodel and allocates it to the tasks that comprise the Work Breakdown Structure (WBS) for the project. This submodel also allocates effort by labor category across life cycle phases. The life cycle submodel calculates maintenance effort and cost for the system based on the effort selected in the risk submodel. This submodel contains a load-balancing feature that allows a user to assess the impact of fixed workforce on maintenance. Three other models marketed by RCI are required for non-object oriented and non-reuse software development estimates: SoftCost-R is used for estimating real time software in the traditional development mode, SSM is an analogy software sizing tool and Asset-R is a function point sizing tool.

### 19B.8.1 SoftCost-OO Inputs

A key input of SoftCost-OO is size, which can be input either as SLOC or function points. If SLOC is used, the minimum, most likely, and maximum SLOC are input for new, reused, and modified components. In addition to size, SOFTCOST-OO has twenty-eight other inputs, or attributes, in four categories. Some of the categories and inputs are similar to those used in COCOMO and REVIC. Most of these inputs require a rating ranging from "1" to "6" with "3" being "nominal" or "no-effect"; however, a few inputs such as number of organizations allow other numerical values. A help screen

for each input assists the user in selecting proper values. The inputs in each category are as follows:

- **Product attributes:** The attributes in this category are application type (avionics, command and control, simulation etc.), number of organizations, system architecture (centralized, distributed, multiple processors etc.), organizational interface complexity, staff and computer resources availability, and security requirements.
- **Process attributes:** The attributes in this category include use of modern development methods, use of tools, tool and environment stability, degree of standardization, scope of support, and use of peer reviews.
- **Product attributes:** The attributes in this category include a technology usage factor, product complexity, requirements volatility, degrees of real-time and optimization, reuse costs and benefits, and database size.
- **Personnel attributes:** The attributes in this category include analyst capability; application, language, methodology, and environment experience; team capability, and number of OO projects completed.

SoftCost-OO also has a “Quick Run” capability where, if a similar project has previously been estimated, it can be recalled and only size is required as an input. Once size has been entered, the project can be analyzed like any other project.

### 19B.8.2 SoftCost-OO Processing

SoftCost-OO is one of the few models for which the mathematical algorithms are completely described in the Reference Manual. The SoftCost-OO equations are:

$$\begin{aligned} \bullet \quad PM &= A_0 * A_1 * A_2 * A_3 * A_4 * (SLOC)^C \\ \bullet \quad M &= B_0 * B_1 * B_2 * (PM)^D \end{aligned}$$

PM is the effort in person-months; M is schedule in months;  $A_0$  and  $B_0$  are calibration constants which depend on the application type input;  $A_1$  is an architectural constant which depends on the system architecture input;  $A_4$  and  $B_2$  are scaling factors which vary with size, degree of reuse, and number of OO projects completed;  $A_3$  is the product of all inputs not used for other equation factors; and C and D are exponents which vary with number of OO projects completed. The *SoftCost-Ada User's Manual* illustrates values assigned to ratings for all model inputs to help the user understand the effect of each on

effort and schedule. The model also has several other equations, as described in the SoftCost-OO Reference Manual.

### 19B.8.3 SoftCost-OO Outputs

SoftCost-OO computes a nominal estimate in person-months of effort and schedule for each project, along with a productivity value and an estimate of average number of personnel required. The model then allows for investigation of the sensitivity of each of the input variables and exploration of alternatives to the nominal estimate or changes in schedule, effort, or confidence level. The model also provides schedule outputs for Gantt and PERT charts.

### 19B.8.4 SoftCost-OO Calibration

The model contains three calibration files; an Ada file, a C++ file, and a generic OO file. The user must select one of these as an input for a model estimate. The user can change these files to reflect his or her environment, but this must be done carefully. RCI plans to develop an on-line calibration capability to work with SoftCost-OO as a separate product.

### 19B.8.5 SoftCost-OO Life Cycle Considerations

SoftCost-OO contains a separate life cycle model for support costs. In addition to SoftCost-OO developmental inputs, life cycle inputs include annual change traffic, length of the support period, a sustaining engineering factor, and economic factors. There is also a provision for entering and reporting on various staff levels, fixed support costs, and fixed work force levels.

## 19B.9 SOFTWARE LIFE CYCLE MODEL (SLIM)

This model was developed by Quantitative Software Management (QSM) Corporation, and is based on the work of Lawrence Putnam. SLIM is proprietary; however, much of the theory behind the model is published in previous works by Putnam, and his book *Measures of Excellence*. A key feature of SLIM is the use of the Rayleigh-Norden curve, illustrated in Figure 19D-1, to allocate resources during a project. The time integral of the Rayleigh-Norden curve results in the “software equation”, which is as follows: (Note:  $T_D$  is development time.)

$$\text{Size} = (\text{Productivity Factor}) * (\text{Effort})^{1/3} * T_D^{4/3}$$

This software equation is fundamental to SLIM and the entire QSM approach.

SLIM is applicable to all types of projects, although it was originally developed for large projects. It computes costs for all software development phases. The “main build” phase initially computed by SLIM includes the detailed design through system test phases, but the model has the option to include the “requirements and design” phase, including software requirements and



preliminary design, and a “feasibility study” phase to encompass system requirements and design.

**Product Description:**

SLIM is a software cost, schedule, risk and reliability estimation tool for planning, control and risk analysis of developing software systems. The SLIM model is a combination of a program evaluation and review technique (PERT) algorithm, linear programming, Monte Carlo simulation and sensitivity profiling techniques. It uses expert system methodology and can be customized to a specific organization through the use of historic data. Associated software tools that can be procured from QSM to augment the capabilities of SLIM are SLIM Control (controls Software projects by employing advanced statistical process control techniques for monthly “health checks” using the SEI recommended “core metrics”) and size planner.

### 19B.9.1 SLIM Input

The significant user inputs to the model are as follows:

- **Size:** Size is one of the two key user inputs to SLIM. Size is usually input as SLOC. SLIM does allow the user to input function points which are converted to SLOC using a ratio, or “gearing factor”, that may be specified by the user. The user may also input the programming language used and, as in REVIC, inputs the least, most likely, and greatest size for each program to be analyzed.
- **Productivity Index (PI):** The other key input, PI, is a calibratable parameter that can either be input directly by the user or is computed by the model based on a multitude of additional inputs. The PI is a number that can vary from 1 to 40; higher values result in lower cost and schedule. It is also a very sensitive parameter; a change in one integer value can result in a twenty-five to thirty percent change in cost. SLIM uses fractions of PIs for fine-tuning. The inputs for computing or adjusting PI are as follows:
  - ⇒ **Application type:** This is the key determinant of PI when the user asks the model to compute it. Some of the nine application types are micro-code, avionics, command and control, telecommunications, and business. If a program is of multiple application types, the user can specify percentages of each type.
  - ⇒ **Tooling and methods:** This set of inputs includes factors such as hardware familiarity, use of various types of automated tools, and robustness and adherence to a development standard. For this set of additional inputs and for the next two sets, the user rates each factor from 1 to 10, with 5 usually being average.

- ⇒ **Technical difficulty:** This set of inputs includes the amount of new algorithms and logic, platform stability, and various complexity factors.
- ⇒ **Personnel profile:** This set of inputs includes management effectiveness, skill level, experience, degree of communication, and morale factors such as motivation and cohesiveness.
- **Other Inputs:** The user can specify other factors that affect cost and schedule, including the following:
  - ⇒ Phase adjustments: The user can specify the staffing profiles to be used during each phase of development, and can customize development phases.
  - ⇒ Importance: The user can state the relative importance of cost, schedule, and quality for the program being estimated.
  - ⇒ Constraints: The user can specify maximum cost or effort, maximum schedule, minimum and maximum staffing levels, and **minimum mean-time-to-defect (MTTD)** for the final product. The user can also specify the desired probability of meeting each constraint. The model will attempt to meet all constraints or determine an “optimal” solution that has the greatest chance of meeting all specified constraints.

### 19B.9.2 SLIM Processing

Although the actual equations for SLIM are proprietary, it is known that SLIM relies heavily on the Rayleigh-Norden curve and its inherent assumptions. The original curve assumed that the maximum manpower was allocated at TD, the development time, and that there was a 60:40 ratio of support costs to development costs. SLIM adjusts this curve as required to meet project-peculiar inputs, especially for smaller programs, and user-specified constraints.

The shape of the curve is determined by three key parameters within the model: size, PI, and the **manpower buildup index (MBI)**. The MBI is a number that varies between -3 and 10. It reflects the rate at which personnel are added to a project. Higher ratings indicate faster buildup rates, and result in shorter schedules but higher costs. The Rayleigh curve, as shown in Figure 19D-1, is shifted upward and to the left as MBI increases. Lower size or higher PI values result in both lower costs and shorter schedules; the Rayleigh curve is shifted downward and to the left. Although MBI is a significant parameter, the user cannot input MBI directly. Instead, it is determined primarily by the user-specified constraints.

### 19B.9.3 SLIM Outputs

The primary output of SLIM is an optimal solution to meet the constraints specified by the user. In the absence of constraints, the model will compute a “minimum time” solution for which the user can reduce cost by relaxing the schedule. The “staffing view” of the model shows a staffing profile along with the model’s computed cost, effort, schedule, peak staffing required, and MTTD. The staffing view also shows the probabilities of meeting each of the specified constraints. The model also has a “ballpark view” and a “consistency view” where time, effort, MTTD, average staffing, and PI are compared with similar projects in the SLIM database. SLIM has numerous report options for risk analysis, defect profiles, and other areas of interest to the user. The model has a very impressive feature that allows the user to vary a certain parameter on a chart with the mouse and see the effect on the other parameters, including risk.

### 19B.9.4 SLIM Calibration

The PI for SLIM can (and should) be calibrated using historical data. The model has on-line calibration capability for the user to calibrate a PI from historical projects. All that is required are program size, development time in months, person man-months of effort, and, if available, MTTD for each historical project. PI is determined to the nearest tenth (e.g., 8.3) in calibration.

### 19B.9.5 SLIM Life Cycle Considerations

SLIM has an optional “maintenance” phase output which uses SLIM development inputs to compute man-months, schedule, and staffing profiles. The user can not specify any support-peculiar inputs except the shape of the curve (Rayleigh, stair step, exponential, straight line, or level load). The model computes all support outputs based on extrapolation of the staffing curve for the time after software development is completed using the user-specified curve shape, and the time to achieve either 99% or 99.9% reliability as specified by the user.

## 19B.10 SYSTEM EVALUATION AND ESTIMATION OF RESOURCES SOFTWARE ESTIMATING MODEL (SEER-SEM)

One of a family of models marketed by Galorath Associates, SEER-SEM is based on the work of Randall Jensen in his paper “An Improved Macro-Level Software Development Resource Estimation Model.” It uses the Rayleigh-Norden curve (described in SLIM) to allocate resources during a software project and to estimate cost and schedule. SEER-SEM is applicable to all types of programs, and is applicable to all phases of the software development cycle except system requirements and design. The model is proprietary.

### 19B.10.1 SEER-SEM INPUTS

The inputs for this model can be divided into three categories: size, knowledge base inputs, and input parameters. Each category of inputs is now summarized.

- **Size:** The user can input size using one of three measures: SLOC, traditional function points, or Galorath function points. SEER converts function points to SLOC before generating an estimate. All code is further categorized as “new”, “preexists designed for reuse”, or “preexists not designed for reuse”. For pre-existing software, the user must specify the amount of code deleted, and the percentages of redesign, recoding, and retest required to modify or reuse the program for the current application. The model uses PERT; therefore, the user must input a “minimum”, “most likely”, and “maximum” value for all size inputs.
- **Knowledge Base Inputs:** SEER-SEM contains a myriad of knowledge bases for different types of software. The knowledge bases assign default values to the input parameters described below based on the type of software selected. The user must specify the following six inputs to specify the knowledge base to be used by the model.
  - ⇒ **Platform:** The operating environment of the program, such as avionics, ground-based, or manned space.
  - ⇒ **Application:** The overall function of the software, such as radar, command and control, mission planning, or testing.
  - ⇒ **Acquisition method:** The method in which the software is to be acquired, such as development, modification, or re-engineering.
  - ⇒ **Development method:** The method used for development, such as waterfall development, evolutionary development, object oriented, prototype or incremental development.
  - ⇒ **Development Standard:** The Standard used in development and degree of tailoring to describe documentation, quality, and test standards such as ISO, 2167A etc.
  - ⇒ **Class:** This input is primarily for user-defined knowledge bases.
- **Input Parameters:** SEER-SEM contains more than thirty input parameters with which the user can refine an estimate. As in COCOMO, the values normally range from “very low” to “very high”. As in size, the user must specify a least, greatest, and most



likely value for each input. A user can use the default values generated by the chosen knowledge base if no further information is available. The primary categories of input parameters and a brief description of each follows.

- ⇒ **Complexity:** Assesses the difficulty of the software.
- ⇒ **Personnel capability and experience:** The parameters in this category, similar to the “personnel attributes” of COCOMO, measure the caliber of personnel used on the project.
- ⇒ **Development support environment:** Measures the usage of modern practices and tools, availability of resources, and frequency of changes to the environment.
- ⇒ **Product development requirements:** Measures the stringency of quality, documentation, and test requirements, as well as the frequency of changes in requirements.
- ⇒ **Reusability requirements:** Measures the degree of reuse needed for future programs and the percentage of software affected by reusability requirements.
- ⇒ **Development environment complexity:** Measures the complexity of the language, application, and host development system used.
- ⇒ **Target environment:** Measures special constraints for the target environment such as memory, special displays, and security (which is the most sensitive input parameter in the model).
- ⇒ **Other input parameters:** There are also special inputs for schedule constraints, labor rates, integration requirements, personnel costs, metrics, and software support.

### 19B.10.2 SEER-SEM Processing

Although the model is proprietary, some of the equations of the SEERSEM model can be found in the *SEER-SEM User's Manual* and in articles published by Randall Jensen. SEER-SEM computes an **effective technology rating (ETR)** based on several input parameters. The model apparently uses the Rayleigh-Norden curve to compute the required effort. SEER-SEM also contains windows where the user can compare two projects, examine several risk analysis graphs, and see what effect a changed input parameter will have on the overall development cost and schedule.

### 19B.10.3 SEER-SEM Outputs

SEER-SEM allows the user to select a variety of output reports and charts. A “quick estimate” provides a snapshot of size effort, schedule, and ETR anytime during the estimating process. Optional outputs include a basic estimate, staffing by month, cost by month, cost by activity, person-months by activity, and software metrics including delivered defects.

### 19B.10.4 SEER-SEM Calibration

The model may be calibrated by computing an ETR from past programs, by computing cost and schedule multipliers from past programs, or both. The cost and schedule multipliers are linear multipliers with a nominal value of one (which would have no effect). The ETR, multipliers, or both can be incorporated into a custom knowledge base for future programs of the type calibrated. One limitation of the ETR is that the user cannot input it directly as a model input; it must be adjusted by changing other input parameters. The cost and schedule multipliers, however, can be input directly. In addition to ETRs, knowledge bases and parameter setting may be calibrated.

### 19B.10.5 SEER-SEM Life Cycle Considerations

SEER-SEM contains an optional “maintenance” output report which provides annual costs and person-months for each year of a user-specified schedule in four categories: corrective, adaptive, perfective, and enhancements. The user can specify the support time period desired along with several other support-unique parameters. These include annual change rate, number of support sites, expected program growth, and differences between development and support personnel and environment, and degree of rigor (level of support).

## 19B.11 OTHER MODELS

The models discussed above do not encompass the entire arena of software parametric cost models; there are numerous other models available. For example, there are several variants of COCOMO available in addition to REVIC. Several companies have software cost models that are used solely within the company which developed the model. New models and new versions are likely to be available.

## Commercial Off-The-Shelf (COTS) Software

### 19C.1 INTRODUCTION

This appendix summarizes the special considerations related to incorporating COTS (commercial off-the-shelf) software components into a system. These considerations vary so much from standard software development approaches that special versions of software cost estimating models are being created to deal with the peculiarities. If a COTS software product is to be used as a stand-alone item, then some subsections such as those considerations related to “glue code” do not apply. For a COTS software product that will be integrated into a system, all the sections of this appendix apply. The majority of this introduction is extended from the model rationale used by the University of Southern California (USC), Center for Software Engineering (under the direction of Dr. Boehm) for the COTS version of COCOMO.

A **stand alone item** does not need interfaces with other software components

One source of pre-existing software is commercial vendors who supply self-contained off-the-shelf components that can be plugged into a larger software system to provide capability that would otherwise have to be custom-built. The two primary distinguishing characteristics of COTS software are 1) that its source code is not available to the application developer (software developer that will incorporate a COTS component into a larger system), and 2) that its evolution is not under the control of the application developer.

The rationale for building COTS based systems is that development time is reduced by taking advantage of existing, market proven, vendor supported products, thereby reducing overall system development costs. In the case of using such components as databases and operating systems, this is almost certainly true. However, there are little data available concerning the relative costs of using the component-based approach. Due to a of the lack of access to product source code and a lack of control over product evolution, there is a trade-off that results using the COTS approach. New software development time can indeed be reduced; however, the cost of software component integration generally increases. Moreover, if integrating COTS components, there will be additional system costs to negotiate, manage, and track licenses to ensure uninterrupted operation of the system. Moreover, using COTS software also brings with it a host of unique risks quite different from those associated with software developed in-house.

The true cost of integrating a COTS software component into a larger system includes the traditional costs associated with new software development such as the cost of requirements definition, design, code, test, and software maintenance. In addition, the cost of integration includes: the cost of

licensing and redistribution rights; royalties; effort needed to understand the COTS software; pre-integration assessment and evaluation; post-integration certification of compliance with mission critical or safety critical requirements; indemnification against faults or damage caused by vendor supplied components; and costs incurred due to incompatibilities with other needed software and/or hardware.

The following sections explain the system development model phases, cost estimation model status, cost estimation/pricing equations, testing/supportability/risk considerations, and licensing schemes for COTS software.

## 19C.2 COTS SOFTWARE FIVE PANEL MODEL

This model was obtained from the Software Engineering Institute (SEI) at Carnegie Mellon University, who have authored a number of Special Reports and hosted Workshops devoted to COTS software.

### 19C.2.1 COTS Market Phase

The COTS Market phase deals with the market survey and analysis activities that determine the viable candidates for a particular component, from both a business and a technical perspective. Discussing the many aspects regarding a market analysis is outside the scope of this appendix.

### 19C.2.2 COTS Qualify Phase

The **Qualify** phase activities investigate the hidden interfaces and other characteristics and features of the candidate products. Per SEI in their article “Component-Based Software Development/COTS Integration,” component qualification is a process of determining “fitness for use” of previously-developed components that are being applied in a new system context. Qualification of a component can also extend to include qualification of the development process used to create and maintain it (for example, ensuring algorithms have been validated, and that rigorous code inspections have taken place).

Continuing with the SEI article, there are two aspects of component qualification: discovery and evaluation. For discovery, the properties of a component are identified. Such priorities include component functionality (what services are provided) and other aspects of a component’s interface (such as the use of standards.) These properties also include quality aspects that are more difficult to isolate, such as component reliability, predictability, and usability. In some circumstances, it is also reasonable to discover “non-technical” component properties, such as the vendor’s market share, past business performance, and process maturity of the component developer’s organization. Discovery is a difficult and ill-defined process, with much of



the needed information being difficult to quantify and, in some cases, difficult to obtain.

The most difficult problem in the Qualify phase is to determine the characteristics of the available COTS products so that a product which best meets the requirements may be selected. The result of this discovery process is to reveal the necessary information to make a selection and identify possible sources of conflict and overlap, so that the component can be effectively assembled and evolved. The following excerpt from a Crosstalk article describes the selection process.

*"Experience shows that the selection process for one major product can require three to six months of calendar time, multiple engineers and programmers, access to sophisticated suites of hardware and software environments, and will likely entail the purchase of vendor-provided training classes."*

*March 1998 STSC Crosstalk*

The rationale behind this time and expense are summarized by Richard D. Stutzke in his paper, "Cost Factors for COTS Integration." Typically the COTS capabilities which are of interest to the designer are the completeness of the functions provided, the COTS product's architecture, and the maturity and expected life of the product. The product's architecture is particularly important since COTS products often make assumptions about the environment in which they will operate. The product's architecture dependencies also affect the partitioning of the components of the COTS product and their dependencies. These dependencies can have significant cost impacts when the various components are integrated.

Dr. Stutzke states that during the evaluation process, the analyst must build a mental model which relates the general and specific knowledge for both the system being built and the COTS product being considered for inclusion into that system. He referred to three models that the analyst must construct to understand a software program:

- Structure ("top-down hierarchy of elements")
- Control flow ("program")
- Data flow ("situation")

A key point concerning these models is that knowledgeable people are essential to correctly evaluate COTS components expeditiously.

### 19C.2.3 COTS Adapt Phase

Because individual components are written to meet different requirements,

and are based on differing assumptions about their context, components often must be adapted when used in a new system. A March 1998 article in STSC Crosstalk on COTS states that COTS software does not require coding but does require integration with other components. As a result, it starts the life cycle as a partially developed component. The design, construction, and integration and test development stages must be recast to accommodate early COTS software integration and testing as well as to develop “Glue Code” (interface software, configuration files, scripts, utilities, and data files) which is required to make the COTS software deliver its intended functionality.

Referring again to the SEI article, components must be adapted based on rules that ensure conflicts among components are minimized. The degree to which a component’s internal structure is accessible suggests different approaches to adaptation:

- **White Box.** Access to source code allows a component to be significantly rewritten to operate with other components.
- **Grey Box.** Source code of a component is not modified but the component provides its own extension language or application programming interface (API).
- **Black Box.** Only a binary executable form of the component is available and there is no extension language or API.

Each of these adaptation approaches has its own positives and negatives; however, White box approaches, because they modify source code, can result in serious maintenance and evolution concerns in the long term. Wrapping, bridging, and mediating (included as part of the “Glue Code”) are specific programming techniques used to adapt Grey- and Black-box components.

#### 19C.2.4 COTS Assembly Phase

The **Assembly** phase is the integration of the adapted components into a software architectural infrastructure. Components must be integrated through some well-defined infrastructure that provides the binding that forms a system from the disparate components. This infrastructure supports component assembly and coordination, and differentiates architectural assembly from ad hoc “glue”. Referring again to the March 1998 STSC Crosstalk article, waiting until late in the development process to test and integrate COTS products, particularly those that are complex, will not give adequate time to master all their intricacies and complexities. COTS product testing and integration activities must be interwoven into more of the development process stages.

Extended from Dr. Stutzke’s paper, Table 19C-1 summarizes two types of potential problems that may occur during the Assembly phase. First, it is very

difficult to determine exactly how hard it will be to tailor and integrate the COTS product into the new system. As indicated in the table, the steps are first to obtain a detailed, understanding of the component, then the component must be modified and tested. In some cases the product must be documented as well.

**Table 19C-1. Managing Potential Assembly Phase Problems**

Predicting how hard it will be to integrate the code
<ul style="list-style-type: none"><li>• Obtain a detail understanding</li><li>• Modify</li><li>• Test</li><li>• Document (if required)</li></ul>
Incorporating new versions during integration
<ul style="list-style-type: none"><li>• Cost</li><li>• Strategy</li></ul>

The second potential problem is that new versions of the COTS products are often released during the integration process. This can affect the cost of development, therefore some sort of strategy is needed to manage this. The likelihood of new versions increases if the project lasts longer than one year.

The particular process used by the development team can significantly affect the cost of integrating COTS components. The three important process areas are installing and integrating the component initially, installing new versions of the components which become available during the integration process, and tracking and controlling the baselines.

### 19C.2.5 COTS Update Phase

The **Update** phase acknowledges that new versions of components will replace older versions; in some cases, components may be replaced by different components with similar behavior and interfaces. These replacement activities may require that glue code be rewritten, and they suggest the advantage of well-defined component interfaces that reduce the extensive testing otherwise needed to ensure that the operation of unchanged components is not adversely affected.

Per the SEI article, component-based systems may seem relatively easy to evolve and upgrade since components are the unit of change. To repair an error, an updated component is swapped for its defective equivalent, treating components as plug-replaceable units. Similarly, when additional functionality is required, it is embodied in a new component that is added to the system. This is a highly simplistic (and optimistic) view of system evolution however.

The new component will never be identical to its predecessor and must be thoroughly tested, both in isolation and in combination with the rest of the system. As a result, replacement of one component with another is often a time-consuming and arduous task. Wrappers must typically be rewritten, and side-effects from changes must be found and assessed.

Dr. Stutzke concluded that the “volatility” of COTS products is a source of significant uncertainty and risk, increasing in magnitude as the length of the system’s planned service life increases. A COTS product is volatile in two ways: features and price. COTS products are market-driven and evolve rapidly in response to consumer and competitive pressures. The vendor adds (and sometimes deletes) features based on many factors. Regarding pricing, maintenance pricing is the most volatile (assuming the initial purchase occurs shortly after the evaluation is completed.)

Changes in maintenance price are primarily of interest for estimating the operating costs of the deployed system. Typically, vendors encourage the user to discontinue use of a product they want to retire by increasing the annual maintenance fees (to cover the reduced number of active users.) They may also offer attractive price reductions on the purchase price of new, replacement products. The obsolescence of product features and/or changes in maintenance price may cause significant redesign and refurbishment of the deployed system long before its useful life is over. Of course, these costs impact the lifecycle costs of the system.

### 19C.3 COTS COST ESTIMATION MODEL STATUS

The USC’s Center for Software Engineering under the direction of Dr Boehm is in the process of developing a version of COCOMO tailored for the unique requirements of estimating COTS software. This model is called COCOTS (CONstructive COTS). COCOTS is considered experimental and evolving. The latest information on this model can be obtained at website <http://sunset.usc.edu/research/COCOTS/modeldesc.html>. The contents of this section were extended from the information obtained from the COCOTS web site.

COCOTS’ goal is the development of a comprehensive COTS integration cost modeling tool. The approach taken was to first examine a wide variety of sources in an attempt to identify the most significant factors driving COTS integration costs, and then to develop a mathematical form for such a model.

#### 19C.3.1 COCOTS Model

The current model provides insight into the most important factors that should be considered when estimating the cost of integrating COTS components, regardless of the specific tool or methodology used to perform that estimation. “The model represents a prototype implementation of the



first of four submodels being proposed for COCOTS, namely, the glue code submodel. The model is not yet mature enough for the estimates it provides to be used with a high level of confidence. However, the cost parameters contained within the model and the criteria used to rate those parameters as described in the user guide certainly offer much insight into the questions a software cost estimator should be considering when working on a system to be built with COTS components.”

The USC COTS integration cost model version 1.0 takes the following form:

$$\text{SIZE} = \text{KSLOC} (1.0 + \text{BRAK}/100)$$

$$\text{PM} = A * (\text{SIZE})^{\beta} * \prod_{i=1}^{13} \text{EM}_i$$

$$\text{COST} = (\text{PM}) * (\$/\text{PM})$$

Table 19C-2. Definition of Symbols

Symbol	Description
A	A linear scaling constant to provide an accurate effort estimate when all EM are nominal, provisionally set to 12.0.
$\beta$	A nonlinear scaling constant that accounts for the influence of factors that have exponential effects, provisionally set to 1.0.
BRAK	Breakage: Percentage of COTS glue code thrown away due to requirements volatility.
KSLOC	Size of COTS component glue code expressed in thousands of source lines of code.
PM	Person Months of estimated COTS integration effort.
\$/PM	Estimated average labor rate per person-month.
EM	The thirteen Effort Multipliers or cost drivers, each of which assumes one of five possible values based upon the following ratings: very low, low, nominal, high, and very high. Nominal ratings have a multiplier value of 1.0.

USC suggests the following procedures when attempting to use its model for a COTS integration cost estimation exercise:

1. Estimate the amount of glue code that is expected to be needed to integrate a set of COTS products into a software application. Only the integration or glue code linking the COTS component to the larger application should be included in the estimate, not the code internal to the COTS component itself.
2. Estimate the percentage of glue code that will be lost due to breakage during the integration effort. This will be a function of the number

of COTS packages being integrated into the new system overall, the average number of updated product releases expected per COTS package over the life of the system development, and the average interface breakage per product release. Breakage refers to COTS integration code that must be reworked as a result of a change in system requirements. Breakage also includes integration code required when a new release by the vendor of a COTS product which necessitates that the newer version of the product be installed before system delivery. Breakage does not refer to code that must be reworked due to bugs introduced by the programmer, or due to defects in design. The Breakage percentage is best estimated by acquiring knowledge of two things: 1) the vendor's past history regarding releases of the COTS product in question or of similar products that the vendor markets, and 2) the customer's past history regarding demanding changes in requirements after development and COTS product integration has begun.

3. Determine the effective size of the glue code development effort by feeding the estimates derived in steps 1 and 2 into the formula for SIZE.
4. Assign each effort multiplier a rating on the given scale from very low to very high, which best characterizes the unique conditions pertaining to the COTS integration effort.
5. Determine the overall estimated effort for this integration task by feeding the estimate for SIZE and the rated effort multipliers into the formula for Person-months (PM).
6. Determine the estimated cost by multiplying estimated PM by the estimated average labor rate (\$\$/PM).

This completes the COTS integration cost estimation procedure.

### 19C.3.2 Other COTS Cost Models

There are other COTS Cost Models that are not addressed here.

## 19C.4 COTS COST ESTIMATION/PRICING EQUATIONS

Estimating the cost of a software system that includes COTS components requires the inclusion of more cost areas than typical software development efforts. Figure 19C-2 provides a general illustration of the standard software development costs without COTS. Figure 19C-3 shows the four additional cost areas when COTS components are integrated to form a complete system.

Figure 19C-2. SW Cost Estimates without COTS

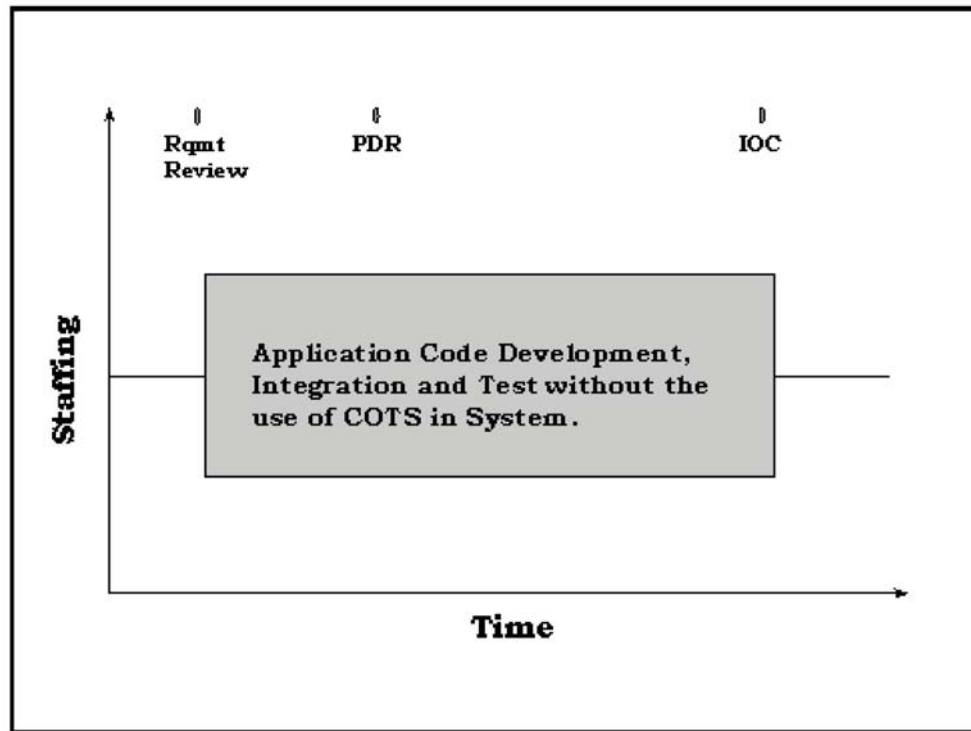
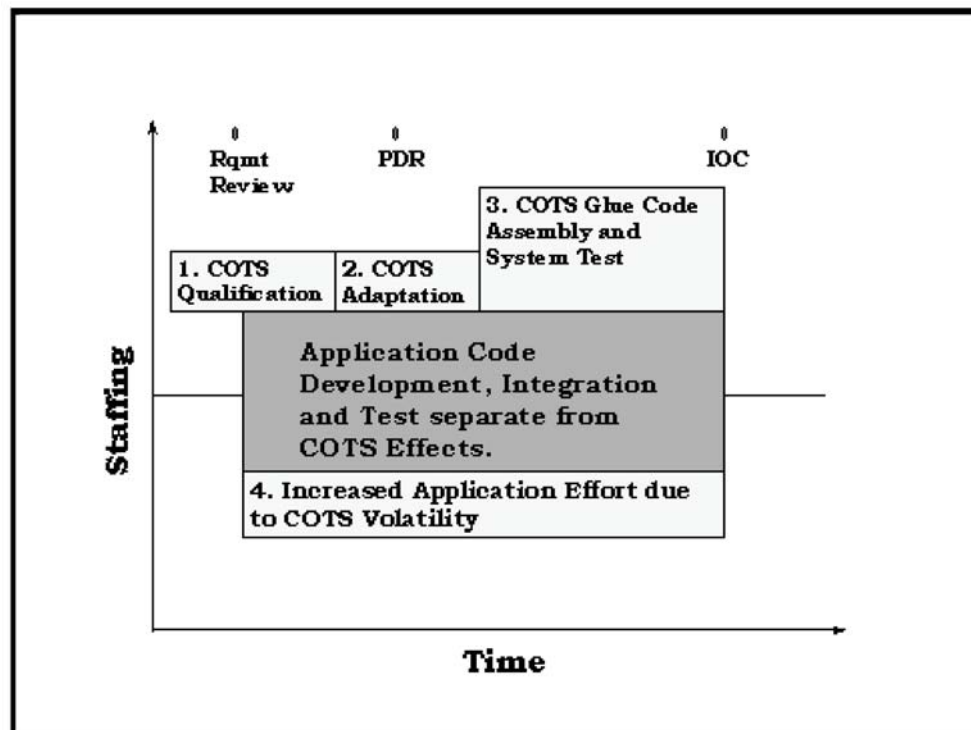


Figure 19C-3. SW Cost Estimates with COTS in System



### 19C.4.1 Software System with COTS Components Costs

The total cost of a software system that includes COTS components could be estimated by adding together the following:

Table 19C-3 Total Cost of Software System

Cost Components	Cost Estimation Model
New SW development (not COTS)	SW cost models in Appendix D
New SW maintenance (not COTS)	SW cost models in Appendix D
COTS SW integration	COCOTS
COTS SW integration maintenance	COCOTS
COTS SW	See below
COTS HW	See below

Estimating the cost of obtaining the COTS software for each COTS component would take the following form:

$$\begin{aligned}
 \text{COTS SW} = & \text{Cost}[\text{acquisition office}] + \text{Cost}[\text{licenses}] \\
 & \{= \text{Cost}[\# \text{ licenses}, \# \text{ features}, \# \text{ platforms}]\} + \\
 & \text{Cost}[\text{implementation}] \{= \text{Cost}[\text{training}] + \text{Cost}[\text{install}]\} + \\
 & \text{Cost}[\text{Op. \& Maint.}] \{= \text{Cost}[\text{maint. licenses}] + \text{Cost}[\text{support}]\}
 \end{aligned}$$

Estimating the cost of obtaining the COTS Hardware would take the following form:

$$\begin{aligned}
 \text{COTS HW} = & \text{Cost}[\text{acquisition office}] + \{ \text{Cost}[\text{acquisition}]_i + \text{Cost}[\text{implementation}]_i \\
 & + \text{Cost}[\text{O \& M}]_i \}
 \end{aligned}$$

*i = number of processors, storage, workstation, communications*

Unit costs vary by quantity, platform, and time. If all the items are not procured at the same time, there will be a need to consider time-phasing of acquisition, implementation, operations & maintenance. The biggest challenge will be the complex, dynamic COTS price structures.

### 19C.5 IMPACT OF COTS ON SYSTEM TESTING

This section is included in order that the analyst may consider these two questions when pricing a proposal or developing a cost estimate. Does the proposal/estimate cover these tests? Is some testing missing or is there too much testing? An important advantage of COTS software is reduced delivery time. This is realized through the elimination of the research and development phase of the acquisition cycle and subsequent development test



and evaluation. In general, testing is not required when the software will be used and maintained by similarly skilled people in the same environment as that for which it was designed and existing data (contractor or other sources) provides reasonable answers to performance and supportability issues. If COTS software components are being integrated into a larger system, then system integration testing and selected component testing would be required. Operational testing is strongly recommended if operational or support environments differ from the contractor's. Further, preselection testing in the Qualify phase should be conducted to minimize technical, operating and support risk.

For stand-alone COTS in a commercial-like (same-environment) application, no testing should be required if the application is the same as commercial use of the item. If there is any doubt, an analyst should acquire and qualify a test sample prior to selection. After delivery of a successful item, any deficiencies that appear should be covered by the vendor's commercial warranty.

For COTS embedded in a larger system, feasibility testing to qualify a test sample should be done prior to selection and integration into the system. Development Test and Evaluation (DT&E) of the complete system is required. Hardware and software integration tests should be conducted as well as user testing (typically, operational test and evaluation (OT&E)) at the system level. COTS deficiencies that appear during or after system-level testing should be covered by the vendor's commercial warranty. Special considerations for specific testing areas are as follows:

**Development Testing.** Development testing should be conducted at the system level but not at the COTS assembly level since COTS is already developed. If an item does not meet performance standards/requirements, select other COTS or go for a non-COTS design effort.

**Operational Testing.** Selection of COTS items does not automatically mean there will be no need for operational testing. However, operational testing for COTS components (or stand-alone COTS) should be limited or waived if market investigation data and, as applicable, pre-selection qualification testing will satisfy the requirements. Operational testing, as previously mentioned, should be conducted for a system with integrated COTS components.

**Supportability Evaluation.** The impact of COTS on system supportability probably requires testing that should include reliability testing, organizational level maintainability testing, human factors engineering tests, standardization measurements and safety analysis.

**Compatibility Testing.** Follow-on compatibility testing in fielded systems could be conducted for all significant hardware or software revisions to a

COTS item. This testing should verify that the interfaces are not violated, and be completed before the Government accepts the change. Reviewing the vendor's service bulletins and engineering change orders (ECOs) can help identify which changes would require testing.

**Quality Assurance Provisions.** A quality assurance provision should be specified for each system-level functional and physical requirement in the contract. However, quality control of individual COTS items should be covered by the vendor's commercial warranties.

### 19C.6 COTS SUPPORT

The rapidly changing market and technology of COTS items means that it will have a limited useful life cycle. Support requirements should be defined before SIR release. Support requirements should also include an up-front definition of the system support requirements to the item level, a lifetime support strategy and appropriate contract language to implement the support strategy. Acquisition contracts should include support such as maintenance, support equipment and training for the total expected life cycle as well as pre-planned product improvements or when replacements are needed.

A market analysis should be performed while the requirements document is still in draft form. If COTS items are to be embedded in a larger system by a prime contractor, be sure that the prime contractor acquires all the required information from the proposed vendors.

The preferred method of support for COTS products is through local acquisition and locally-acquired contractor service. Any decision for life-cycle contractor logistics support must be accompanied by adequate planning. In the absence of a formal contractor support agreement, plan for an appropriate level of organic management and support for COTS assets.

When determining the support for a commercial item, it is recommended to evaluate the original contractor's support, alternate vendors' support, and Government support with regards to impact on competition, operational requirements, total support costs, and support availability.

Three major areas of concern in COTS support are configuration management and data; maintenance policy; and sustainability.

#### 19C.6.1 Configuration Management Data

Except under very unusual circumstances, it is not wise to demand full design disclosure engineering data for system support. Even if the vendor was agreeable to selling the data, it would quickly be obsolete because the government would not have design control. Contractors can unilaterally

change the design as necessary to suit their customer or market requirements. COTS designs should be documented to the system interfaces level.

For COTS software, the government should acquire and maintain appropriate licensing and subscription services (vendor field change orders and software releases) throughout the life of the system. If the Government secures change authority, it must clearly define the limits of what the Government can change, and restrict alterations in any way that would void the licensing or subscription service. Independent software changes by the Government, or a Government freeze to an earlier COTS software version, will turn COTS into a unique product.

### **19C.6.2 Maintenance Policy**

Employment of COTS will usually lock the government into two-level maintenance: organizational and depot. The most desirable support concept is contract repair, preferably via competition. A good strategy is to price support options while the original buy is still in competition. The riskiest course is an in-house depot repair, which should be selected only under exceptional conditions because it will have to be a specialized operation if even possible at all. The vendor might refuse to sell the data and tools to develop a capability.

### **19C.6.3 Sustainability**

Because of the design control the Government has had over in-house developmental software, many systems have been maintained and upgraded to serve multiples of their original design life. Market pressures usually send commercial designs more quickly into obsolescence not so easily dealt with. It is not uncommon for this to occur within five years, which means that retrofit funding should be routinely projected approximately at the time of original system fielding. There should be cost estimates for maintenance and eventual replacement of COTS software. Licensing and subscription service represent a significant annual cost, and replacement cost will also be considerable.

## **19C.7 LICENSING SCHEMES**

Referring again to Dr. Stutzke's article, "Cost Factors for COTS Integration," the license terms for the COTS products can affect the cost of the overall system and even affect the choice of the architecture. Dr. Stutzke adapted a memo by Gibson and Mankofsky which grouped their license types into the six types shown in Table 19C-4. The names used were defined to be as descriptive as possible since there is no standard terminology used by vendors to refer to these types of licenses. This makes it extremely difficult to compare the costs of products offered by different vendors.



Table 19C-4. Types of COTS Software Licenses

Universal Site
<ul style="list-style-type: none"> <li>Covers use on all machines at a geographic location</li> <li>Annual (Payment of maintenance fee provides yearly update)</li> <li>Perpetual (No maintenance fee. Repurchase required.)</li> <li>Number of users is unlimited</li> </ul>
Site
<ul style="list-style-type: none"> <li>Covers one machine/platform type at a given location</li> <li>Number of users is unlimited</li> <li>May be annual or perpetual</li> </ul>
Node-Locked
<ul style="list-style-type: none"> <li>Covers use on a single machine (tied to machine's serial number)</li> <li>May be annual or perpetual</li> </ul>
Floating
<ul style="list-style-type: none"> <li>Allows multiple users on a client/server network</li> <li>Controlled by one or more "license servers"</li> <li>Buy seats ("sockets") or tokens ("resource units tied to platform type")</li> </ul>
Metered
<ul style="list-style-type: none"> <li>Usage measured and charged (or limited to some specified amount per day, month etc.</li> <li>Can be combined with node locked, floating, platform or token schemes</li> </ul>
Public Domain
<ul style="list-style-type: none"> <li>Pay distribution charge</li> <li>May pay registration fee or donation ("shareware")</li> <li>May be free to non-profit organization or Government agency. (Must purchase the rights for commercial resale)</li> </ul>

The license costs are primarily of interest in making the "buy or build" decision during the Product Design phase or the Proposal phase (since this is when the price must be determined). The cost of developing the products must be traded against the cost of purchasing the necessary number of copies and integrating them into the system. In-house developed products have a significant non-recurring engineering cost which is paid once. All subsequent copies of the component may; however, be used for essentially no cost for the life of the system, regardless of how many copies of the product are produced. For a COTS product, the non-recurring engineering cost is absorbed by the vendor but the buyer (developer of the new system) must pay a license fee for each copy of the component which is used. Complex trades-off analysis must be made between functionality, cost, schedule, risk, etc., involving not just the development and production cost but also the operating costs of the deployed system when performing a make (develop) or buy (COTS) decision.



Other factors may affect the cost of a COTS product. First, discounts may be available if multiple copies are purchased (volume discounts). Second, reduced prices may be offered if several related products are purchased together (“bundling”). Third, sometimes the first year of maintenance is provided free with the initial purchase of the product. Fourth, if the product is utilized during the development process, it is possible that an additional warranty must be purchased prior to delivery of the system to the customer. Fifth, there may be liability costs in the event COTS components cause failure in the system during operations. Such costs could cover damage to equipment, injuries to personnel and loss of service. Last, there may be refurbishment costs associated with the loss of critical functions as the product evolves in response to market pressures. The functions needed may be removed from later versions of the product.

Multiple types of licenses must be mixed to obtain the necessary number of copies of a product. Sometimes a full license is purchased to obtain a complete set of documentation and several other “right to use” licenses are purchased to allow copies of the “full” product to be used on other platforms. Such mixing and matching can have significant impacts on the total cost of a system.

### 19C.8 SUMMARY

The utilization of COTS software components/products is intended to reduce the cost and schedule of software development programs. If the COTS component is a stand-alone product, it probably will. However, for integrating COTS components into a larger system, there are many factors that must be considered. This appendix has outlined the primary considerations and the basic cost equations that apply to COTS software cost estimating and pricing.

## Works Cited

Boehm, Barry W. Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall, 1981.

Boehm, Barry W. and Christopher M. Abts. "COTS/NDI Software Integration Cost estimation & USC-CSE COTS Integration Cost Calculator v2.0 User Guide." Los Angeles, CA: University of Southern California, September 1997.

Boehm, Barry W., et al. "COCOMO 2.0 Program." Presentation handouts. ISPA 1994 Conference, Munich, Germany, 1994.

Boehm, Barry W., et al. "The COCOMO 2.0 Software Cost Estimation Model." Abstract. International Society of Parametric Analysts, 1995.

Boehm, Barry W., et al. "An Overview of the COCOMO 2.0 Software Cost Model." Abstract. Software Technology Conference, 1995.

Carney, David J. and John Foreman. "Component-Based Software Development/COTS Integration." Software Technology Review Search Results ([http://www.sei.cmu.edu/str/descriptions/cbsd\\_body.html](http://www.sei.cmu.edu/str/descriptions/cbsd_body.html)): Carnegie Mellon University, 1998.

Fox, Greg, et al. "A Software Development Process for COTS-Based Information System Infrastructure: Part 1." Crosstalk, The Journal of Defense Software Engineering Vol 11, No. 3, March 1998.

Galorath Associates. SEER-SEM User's Manual. Los Angeles, CA: Galorath Associates, September 1994.

Galorath Associates. SEER-SEM User's Manual Release 4.5. Updated. El Segundo, CA: Galorath Associates, September, 1996.

Galorath Associates. SEER-SSM User's Guide. Marina del Rey, CA: Galorath Associates, 1991.

Jensen, Randall W. "An Improved Macro-Level Software Development Resource Estimation Model." Proceedings of the Fourteenth Asilomar Conference on Circuits, Systems, and Computers 1981.

Jones, Capers. "Programming Languages Table." Burlington, MA: Software Productivity Research, March 1995.

Jones, Capers. "What Are Function Points." Burlington, MA: Software Productivity Research, March, 1995.

Jones, Capers. Applied Software Measurement. New York: McGraw-Hill, 1991.

- Jones, Capers. *Programming Productivity*. New York: McGraw-Hill, 1986.
- Musa, John D. et. Al. *Software Reliability: Measurement, Prediction, Application*. New York: McGraw-Hill, 1987.
- PRICE Systems. *The Central Equations of the PRICE Software Cost Model*. Moorestown, NJ: PRICE Systems, 1988.
- PRICE Systems. *PRICE-S Reference Manual*. 3d ed. Moorestown, NJ: Martin-Marietta, October 1993.
- Putnam, Lawrence H., and Ware Myers. *Measures of Excellence*. Englewood Cliffs, NJ: Prentice-Hall, 1992.
- Quantitative Software Management Corporation. *SLIM 3.0 For Windows User's Manual*. McLean, VA: Quantitative Software Management, 1993.
- Reifer, Donald J. *SoftCost-Ada User's Manual, Version 2.2*. Torrance, CA, Reifer Consultants, Inc., 1991.
- Reifer, Donald J. *SoftCost-R User's Manual Version 8.0*. Torrance, CA: Reifer Consultants, Inc., 1989.
- Resource Calculations, Inc. *SoftCost-OO User's Guide Version 3.1*. Denver, CO: Resource Calculations, Inc., 1994.
- Roetzheim, William H., and Reyna A. Beasley. "Best Practice in Software Project Cost and Schedule Estimating." *Cost Xpert User's Manual*. Jamul, CA: Marotz, Inc., 1997.
- Software Engineering Institute. *Checklists and Criteria for Evaluating the Cost and Schedule Estimating Capabilities of Software Organizations*, CMU/SEI-95-SR-005. Pittsburgh, PA: Carnegie Mellon Univ., January 1995.
- Software Engineering Institute. *Workshop on COTS-Based Systems*, CMU/SEI-97-SR-019. Pittsburgh, PA: Carnegie Mellon Univ., November 1997.
- Software Productivity Research, Inc. *KnowledgePLAN User's Guide Version 2.0*. Burlington, MA: Software Productivity Research, April 1997.
- Stutzke, Richard D. "Cost Factors for COTS Integration", *Proceedings of the 10th International COCOMO User's Conference*. October 1995.
- United States. U. S. Air Force Material Command "White Paper #2 on Methods for Evaluating Similar Items." Dayton, OH: AFMC, January 1996.
- United States. U. S. Naval Center for Cost Analysis. *Software Development Estimating Handbook*. Arlington, VA: NCCA, February 1998.

United States. National Aeronautics and Space Administration. Parametric Cost Estimating Handbook. Springfield, VA: NTIS, Fall 1995.

United States. National Aeronautics and Space Administration. Handbook for Software Cost Estimation. Pasadena, CA: JPL, May 2003.

United States. Software Engineering Laboratory. Cost and Schedule Estimation Study Report (SEL-93-002). Greenbelt, MD: SEL, November 1993.

