Modern Empirical Cost and Schedule Estimation Tools

A DACS State-of-the-Art Report

Contract Number F30602-89-C-0082
(Data & Analysis Center for Software)

Prepared for:
Air Force Research Laboratory -
Information Directorate (AFRL/IF)
525 Brooks Road
Rome, NY 13441-4505

Prepared by:
Thomas McGibbon
DoD Data & Analysis Center for Software (DACS)
ITT Industries - Systems Division
Griffiss Business & Technology Park
775 Daedalian Drive
Rome, NY 13441-4909

DoD Data & Analysis Center for Software (DACS)
P.O. Box 1400
Rome, NY 13442-1400
(315) 334-4905, (315) 334-4964 - Fax
cust-laisn@dacs.dtic.mil
http://www.dacs.dtic.mil

The Data & Analysis Center for Software (DACS) is a Department of Defense (DoD) Information Analysis Center (IAC), administratively managed by the Defense Technical Information Center (DTIC) under the DoD IAC Program. The DACS is technically managed by Air Force Research Laboratory Information Directorate (AFRL/IF) Rome Research Site. ITT Industries - Systems Division manages and operates the DACS, serving as a source for current, readily available data and information concerning software engineering and software technology.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	20 August 1997	N/A

4. TITLE AND SUBTITLE
A State-of-the-Art-Report
Modern Empirical Cost and Schedule Estimation Tools

5. FUNDING NUMBERS
F30602-89-C-0082

6. AUTHORS
Thomas McGibbon - DACS

7. PERFORMING ORGANIZATIONS NAME(S) AND ADDRESS(ES)
ITT Industries, Systems Division, 775 Daedalian Drive
Rome, NY 13441-4909

8. PERFORMING ORGANIZATION REPORT NUMBER
DACS-SOAR-97-1

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Technical Information Center (DTIC)/ AI
8725 John J. Kingman Rd., STE 0944, Ft. Belvoir, VA 22060
and Air Force Research Lab/IFTD
525 Brooks Rd., Rome, NY 13440

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
N/A

11. SUPPLEMENTARY NOTES
Available from: DoD Data & Analysis Center for Software (DACS)
775 Daedalian Drive, Rome, NY 13441-4909

12a. DISTRIBUTION/ AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

12b. DISTRIBUTION CODE
UL

13. ABSTRACT *(Maximum 200 words)*

Cost models were derived from the collection and analysis of large collections of project data. Modelers would fit a curve to the data and analyze those parameters that affected the curve. Early models applied to custom-developed software systems. New software development philosophies and technologies have emerged in the 1980s and 1990s to reduce development costs and improve quality of software products. These new approaches frequently involve the use of Commercial-Off-The-Shelf (COTS) software, software reuse, application generators, and fourth generation languages. The purpose of this paper is to identify, discuss, compare and contrast software cost estimating models and tools that address these modern philosophies

14. SUBJECT TERMS
Empirical Data, Datasets, Cost Estimation, Software Tools, Metrics Software Measurement

15. NUMBER OF PAGES
21

16. PRICE CODE
N/A

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Modern Empirical Cost and Schedule Estimation Tools

Table of Contents

Tables Referenced in Document

Figures Referenced in Document

Modern Empirical Cost and Schedule Estimation Tools

Abstract and Ordering Information

Abstract

Cost models were derived from the collection and analysis of large collections of project data. Modelers would fit a curve to the data and analyze those parameters that affected the curve. Early models applied to custom-developed software systems. New software development philosophies and technologies have emerged in the 1980s and 1990s to reduce development costs and improve quality of software products. These new approaches frequently involve the use of Commercial-Off-The-Shelf (COTS) software, software reuse, application generators, and fourth generation languages. The purpose of this paper is to identify, discuss, compare and contrast software cost estimating models and tools that address these modern philosophies.

Ordering Information:

A bound version of this report, is avaliable for $30 from the DACS Product Orderform or you may order it by contacting:

DACS Customer Liaison

775 Dadaelian Drive

Griffiss Business Park

Rome, NY 13441-4909

(315) 334-4905; Fax: (315) 334-4964;

cust-liasn@dacs.dtic.mil

Acknowledgements:

The author would like to gratefully acknowledge comments on an earlier draft by Mr. Robert Vienneau and the help of Mr. Lon R. Dean in producing this report.

1. Executive Summary

Project managers and project planners are often expected to provide to their management estimates of schedule length and costs for upcoming projects or products. Anyone that has tried to construct and justify such an estimate for a software development project of any size knows that it can be an art to achieve reasonable levels of accuracy and consistency. Empirically-based cost estimation models supporting management needs began to appear in the literature during the 1970s and 1980s. Support services for some of these models also became commercially available.

Cost models were derived from the collection and analysis of large collections of project data. Modelers would fit a curve to the data and analyze those parameters that affected the curve. Early models applied to custom-developed software systems. New software development philosophies and technologies have emerged in the 1980s and 1990s to reduce development costs and improve quality of software products. These new approaches frequently involve the use of Commercial-Off-The-Shelf (COTS) software, software reuse, application generators, and fourth generation languages. The purpose of this paper is to identify, discuss, compare and contrast software cost estimating models and tools that address these modern philosophies.

In evaluating and selecting an estimation tool, or estimation tool provider, one should evaluate:

- The openness of the underlying model in the tool
- The platform requirements of the tool
- The data required as input to the tool
- The output of the tool
- The accuracy of the estimates provided by the model

The openness of the tool developer in providing details of the underlying model and parameters is one attribute distinguishing some models from others. Volumes of information have been published about the models underlying some tools. The vendors of other tools keep information about their model as proprietary and require the user to purchase consulting services to exercise the model. This may not be a practical problem since beginners will most likely hire consulting services anyways, especially if they are not a statistician or modeler. But, since many of the models can be implemented with a simple spreadsheet, being able to understand the “guts” of the model may be important so as to allow one to enhance the model to meet specific needs.

Several cost models have been implemented in commercial toolsets. Those considering purchasing these products need to consider the platform on which the product is implemented. Most commercial cost modeling tools are only available on Windows-based platforms. You will have to determine if your hardware meets minimal requirements for the toolset.

Major factors in selecting a model include assessing the number and complexity of required input parameters, whether default values can be easily and understandably assigned to these parameters, and whether sufficient historical and project data exists within one’s organization and project to feed the model. Many of the models are very flexible in modeling specific situations. To achieve this flexibility, model providers provide numerous parameters to handle multiple situations. A substantial amount of time may be required to initially set-up a model - one must either answer a number of questions or collect significant amounts of historical data on similar projects. If an established measurement program exists within an organization, assembling the required data may not be too difficult. Most models provide ways

to refine estimates as one gains experience with the model. One beneficial side-effect of cost models is that they identify measures to collect and manage in an organization.

Evaluate the outputs and reports from the model. All software cost models estimate project costs, effort, and nominal schedule for a project. Determine whether other data not estimated by the model (such as travel expenses, overhead costs, etc.) can be easily integrated into the reports being generated. Some models provide additional management guidance in their reporting structure. For example, some models provide staffing profile guidance and determine optimum staff size for a project.

Perhaps the most important feature of any model is the accuracy of the estimate provided. One should try to collect as much data as possible from completed software projects. One can determine with this data whether a model would have accurately estimated those projects and how to fine-tune the model for a given environment. If a model gives wildly inacurate or inconsistent results, one should consider other models.

2. World Wide Web Resources

*** The Constructive Cost Model (COCOMO)** is a well-known model used in software cost and schedule estimation. It is a non-proprietary model introduced by Barry W. Boehm in 1981.

COCOMO Project Homepage <http://sunset.usc.edu/COCOMOII/Cocomo.html>

> The COCOMO 2.0 model is an update of COCOMO 1981 to address software development practices in the 1990s and 2000s. It is being developed by USC-CSE, UC Irvine, and 29 affiliate organizations.

REVIC Software Estimation Model <http://sepo.nosc.mil/REVIC.html>

> Includes a downloadable version of Revised Intermediate COCOMO (REVIC) and pointers to more information about software cost modeling.

Softstar Systems <http://www.SoftstarSystems.com/>

> Developers of Costar, an automated implementation of COCOMO.

*** Function Points** provide a unit of measure for software size using logical functional terms readily understood by business owners and users.

International Function Point Users Group <http://www.bannister.com/ifpug/home/docs/ifpughome.html>

> IFPUG is a membership-governed, non-profit organization committed to increasing the effectiveness of its members' information technology environments through the application of Function Point Analysis (FPA) and other software measurement techniques.

Function Point FAQ <http://ourworld.compuserve.com/homepages/softcomp/fpfaq.htm>

> A comprehensive Function Points FAQ, edited by Ray Boehm of Software Composition Technologies

Metre v2.3 <http://www.lysator.liu.se/c/metre-v2-3.html>

> Metre is a freely distributable ANSI/ISO Standard C parser. Reports Halstead metrics, various line and statement counts, backfired Function Points, control depth, identifier count, number of functions and modules, and a call graph.

Programs for C Source Code Metrics <http://www.qucis.queensu.ca/Software-Engineering/Cmetrics.html>

Some free programs to count lines of code, cyclomatic complexity, Halstead metrics, backfired Function Points, etc. for C code. The tools can be compiled on SunOS.

*** Software Cost Estimating Tools**

Cost.Xpert <http://www.marotz.com/CXCopy.html>

Cost.Xpert, produced by Martoz, Incorporated, provides a step-by-step approach to defining a project's cost and schedule. It is backed by years of research and comprehensive historical databases. Cost.Xpert is an easy-to-use software costing tool with the features and sophistication typical of more expensive programs. It supports all popular forms of cost estimating including COCOMO and Function Points.

PRICE Systems <http://www.pricesystems.com>

PRICE-S is a well-known software cost estimating model. Quantitative Software Management <http://www.qsm.com/> Lawrence Putnam is the president of QSM. They distribute SLIM and related software cost modeling tools.

Resource Calculations, Inc. <http://www.rcinc.com/>

RCI distributes sizing and cost models, including ASSET-R, SSM, and

SOFTCOST-R. Software Productivity Research Information Center <http://www.spr.com/>

A leading provider of software measurement, assessment, and estimation products and services. Capers Jones is SPR's chairman and founder.

*** Other Web Resources**

NASA Johnson Space Center (JSC) Cost Estimating Group <http://www.jsc.nasa.gov/bu2/>

Includes a reference manual for parametric cost estimation. This group supports JSC directorates and program offices with parametric cost estimates, trade studies, schedule analyses, cost-risk analyses, and cost phasing. This includes economic analyses of alternative investments, if applicable. They prepare and document cost estimates. They present and defend cost estimates to senior management and review panels.

Project Management and Cost Estimation Training <http://www.stsc.hill.af.mil/pns/pnsindex.html#cost>

STSC Project Management workshops designed to teach the concepts, principles, and practices of effective project management to those who are responsible for the management of software development or maintenance activities.

The International Society of Parametric Analysts <http://mijuno.larc.nasa.gov/dfc/societies/Ispa.html>

The foundation of ISPA is parametric estimating: a cost-effective approach to consistent, credible, traceable, and timely assessment of resource requirements for development, production, construction, and operation of hardware and software projects.

3. Comparison of Products

Software cost models are used to estimate the amount of effort and calendar time required to develop a software product or system. Effort can be converted into cost by calculating the product of an average labor rate and the effort estimate. Detailed evaluation of the underlying mathematics is one way of comparing various models. Table 1 provides equations for estimating effort (in staff months) with several models, as well as a brief explanation of model parameters. Figures 1 and 2 graph these effort equations, thereby illustrating the variation between models in effort estimates. Figure 1 shows the equations for the different development modes for the Basic and Intermediate COCOMO models. The effort multipliers were set to nominal values in graphing Intermediate COCOMO estimates. Variations between Basic and Intermediate COCOMO would result from the average software project differing from nominal COCOMO cost drivers. The COCOMO 2 effort equation is graphed in Figure 2 with nominal effort multipliers and scale factors, no code discarded due to requirements volatility, and no adapted code. Putnam's simplified SLIM model is graphed with an intermediate productivity parameter, suitable for systems and scientific software. Notice that the Walston-Felix model is the only one that shows software development exhibiting increasing returns to scale in which the cost of a larger system is increased less than proportionately. Table 2 shows the schedule estimates (in calendar months) for each model. Each model is described in turn after Table 2.

Table shown on next page.

Table 1: Effort Equations

Model Name	Effort Equation	Parameters
Basic COCOMO	$Effort = a\ (KSLOC)^{b}$	Organic: a=2.4, b=1.05 Semidetached: a=3.0, b=1.12 Embedded: a=3.6, b=1.20
Intermediate COCOMO	$Effort = EAF\ a\ (KSLOC)^{b}$	EAF is the product of 15 cost driver attributes Organic: a=3.2, b=1.05 Semidetached: a=3.0, b=1.12 Embedded: a=2.8, b=1.20
COCOMO 2.0 Application Composition Model	$Effort = \frac{NOP}{PROD}$ $NOP = OP\ \frac{(100 - \%\ reuse)}{100}$	NOP is New Objects Points PROD is Productivity Rate $PROD = \begin{cases} 4 & \text{Very Low} \\ 7 & \text{Low} \\ 13 & \text{Normal} \\ 25 & \text{High} \\ 50 & \text{Very High} \end{cases}$
COCOMO 2.0 Early Design and Post Architecture Model	$Effort = \frac{ASLOC^{(\frac{AT}{100})}}{ATPROD} + a\left[size\ (1+\frac{BRAK}{100})\right]^{b} \pi_{EM_i}$ $b = 1.01 + \frac{1}{100}\sum SF_j$ $Size = KSLOC + KASLOC\ (\frac{100 - AT}{100})\ AAM$	a = 2.5 SF_j = scale factor EM_i = effort multiplier BRAK = percentage code discarded due to requirements volatility ASLOC = size of adapted components AT = percent of components adapted ATPROD = Automatic Translation Productivity AAM = Adaptation Adjustment Multiplier
Putnam's SLIM	Complete Model: $Effort = 12^{5} B\ (\frac{SLOC}{P})^{3} \frac{1}{Schedule^{4}}$ Simplified Model: $Effort = 56.4B\ (\frac{SLOC}{P})^{\frac{9}{7}}$	B is a special skills factor P is a productivity parameter Schedule is the development schedule length (in months)
Walston-Felix	$Effort = 5.2\ (KSLOC)^{0.91}$	
Bailey-Basili	$Effort = 5.5 + 0.73\ (KSLOC)^{1.16}$	
Doty	$Effort = 5.288\ (KSLOC)^{1.047}$	
Albrecht-Gaffney	$Effort = -13.39 + 0.0545\ FP$	FP is the number of function points
Kemerer	$Effort = 60.62 + 7.728 \times 10^{8} FP^{3}$	
Matson, Barret and Meltichamp	$Effort = 585.7 + 15.12\ FP$	

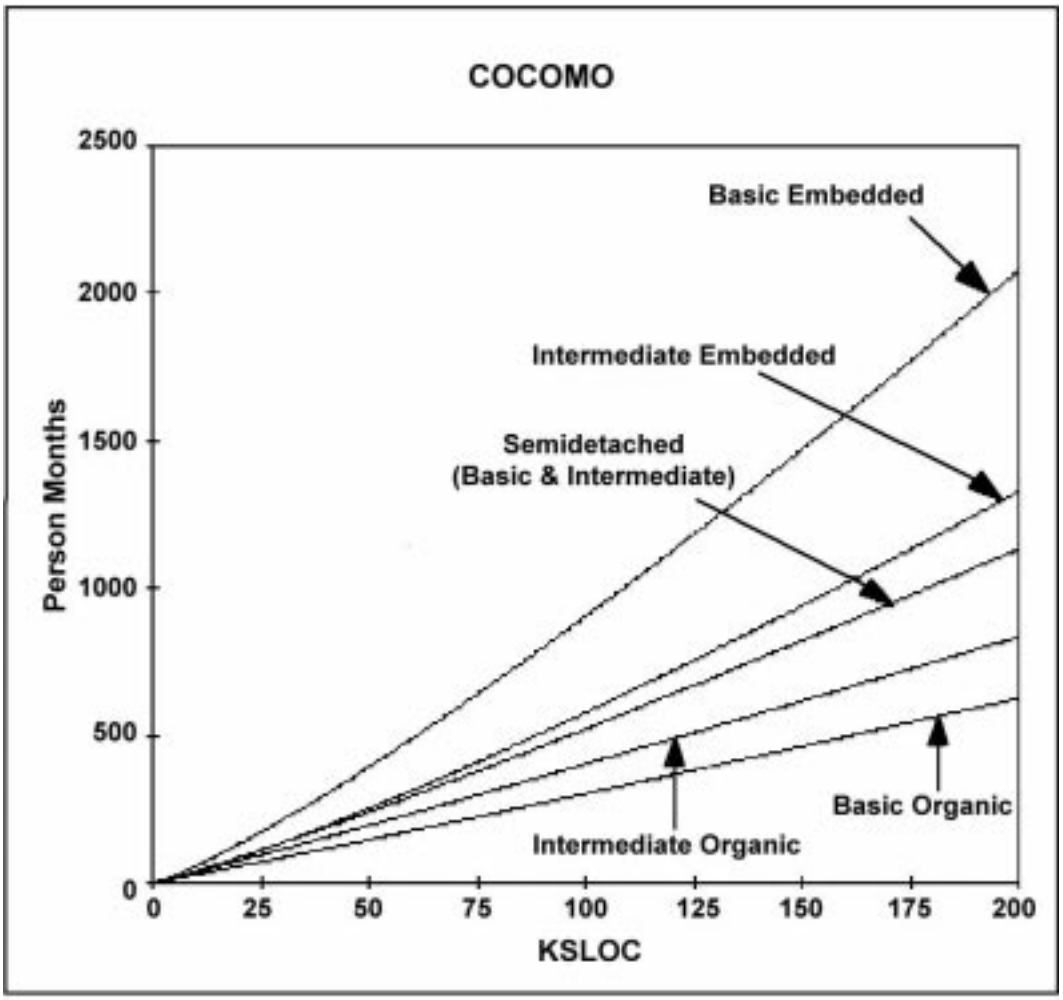

Figure 1: Comparison of COCOMO Effort Equations

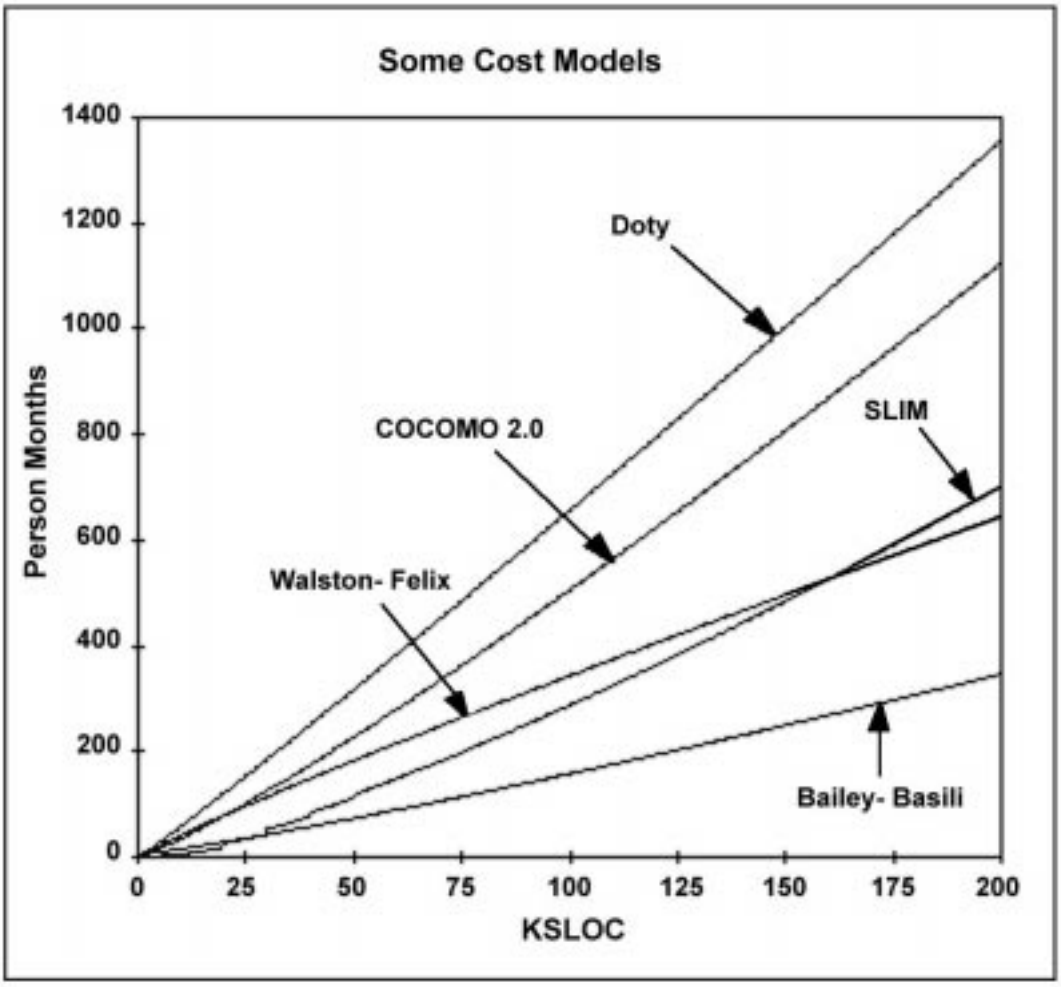

Figure 2: Comparison of Effort Equations

Table 2: Schedule Equations

Model Name	Schedule Equation	Parameters
Basic and Intermediate COCOMO	$\text{Schedule} = 2.5\ \text{Effort}^{C}$	Organic: c = 0.38 Semidetached: c = 0.35 Embedded: c = 0.32
COCOMO	$\text{Schedule} = c\left[\text{Effort}^{0.33+0.2\,(b-1.01)}\right]\frac{\text{SCED\%}}{100}$ $b = 1.01 + \frac{1}{100}\sum SF_j$	c = 3.0 SF_j = scale factor schedule SCED% = compression/ expansion parameter
Putnam's SLIM	Simplified Model: $\text{Effort} = 8.14\left(\frac{\text{SLOC}}{P}\right)^{\frac{3}{7}}$	P is a productivity parameter

Cocomo 1.1

The Constructive Cost Model (COCOMO) is the best known and most popular cost estimation model. COCOMO was developed in the late 1970s and early 1980s by Barry Boehm (1981). This early model consists of a hierarchy of three increasingly detailed models named Basic Cocomo, Intermediate Cocomo and Advanced Cocomo. These models were developed to estimate custom, specification-built software projects. Equations for the Basic and Intermediate models are shown in the first two rows of Tables 1 and 2.

The Basic model expresses development effort strictly as a function of the size (in thousands of source lines of code) and class of software being developed. Organic software projects are fairly small projects made up of teams of people familiar with the application. Semi-detached software projects are systems of medium size and complexity developed by a group of developers of mixed experience levels. Embedded software projects are development projects working under tight hardware, software and operational constraints. The schedule equation, as shown in Table 2, expresses development time (in months) as a function of the effort estimate and the three classes used in estimating effort.

The Intermediate model enhances the effort estimation equation of the Basic model by including 15 cost drivers, known as effort adjustment factors. These 15 factors fall into four categories: product attributes, such as product complexity and required reliability; computer attributes or constraints, such as main storage constraints and execution time constraints; personnel attributes, such as experience with applications and analyst capability; and project attributes that describe whether modern programming practices and software tools are being used.

The Advanced model (not shown in the tables) expands on the Intermediate model by placing cost drivers at each phase of the development life cycle.

Cocomo 2.0

Cocomo 2.0 has recently emerged because of the inability of the original Cocomo (Cocomo 1.1) to accurately estimate object oriented software, software created via spiral or evolutionary models, and software developed from commercial-off-the-shelf software. Several fundamental differences exist between Cocomo 1.1 and 2.0:

- Whereas Cocomo 1.1 requires software size in KSLOC as an input, Cocomo 2.0 provides different effort estimating models based on the stage of development of the project. An objective of 2.0 is to require as inputs to the model only the level of information that is available to the user at that time. As a result, during the earliest conceptual stages of a project, called **Application Composition**, the model uses Object Point estimates to compute effort. Object Points are a measure of software system size, similar to Function Points. During the **Early Design** stages, when little is known about project size or project staff, Unadjusted Function Points are used as an input to the model. Once an architecture has been selected, design and development are ready to begin. This is called the **Post Architecture** stage. At this point, Source Lines Of Code are inputs to the model.
- Whereas Cocomo 1.1 provided point estimates of effort and schedule, Cocomo 2.0 provides likely ranges of estimates that represent one standard deviation around the most likely estimate. (Only the most likely estimates are shown in Tables 1 and 2.)
- Cocomo 2.0 adjusts for software reuse and reengineering where automated tools are used for translation of existing software. Cocomo 1.1 made little accommodation for these factors
- Cocomo 2.0 also accounts for requirements volatility in its estimates.
- The exponent on size in the effort equations in Cocomo 1.1 varies with the development mode. Cocomo 2.0 uses five scale factors to generalize and replace the effects of the development mode.

The Cocomo 2.0 Application Composition model (as shown on the third row of Table 1) uses Object Points to perform estimates. The model assumes the use of Integrated CASE tools for rapid prototyping. Objects include screens, reports and modules in third generation programming languages. Object Points are not necessarily related to objects in Object Oriented Programming. The number of raw objects are estimated, the complexity of each object is estimated, and the weighted total (Object-Point count) is computed. The percentage of reuse and anticipated productivity are also estimated. With this information, the effort estimate shown in Table 1 can be computed.

Function Points are a very popular method of estimating the size of a system, especially early in the life cycle of a product. Function Points are arguably easier to estimate than KSLOC early in the life cycle. Function Points form the size input for the Cocomo 2.0 Early Design model, shown in the fourth row of Table 1. Function Points measure the amount of functionality in a software product based on five characteristics of the product: the number of external inputs, the number of external outputs, the number of internal logical files, the number of external interface files, and the number of external inquiries into the product. Each instance is then weighted by a complexity factor of low, average or high to compute an Unadjusted Function Point (UFP) of:

$$\mathbf{UFP} = \sum_{i=1}^{5} \sum_{j=1}^{3} \mathbf{W}_{ij} \mathbf{Z}_{ij}$$

where $\mathbf{Z}_{ij}$ is the count for component i at complexity j and $\mathbf{W}_{ij}$ is the fixed weight assigned. Unadjusted Function Points are converted to equivalent source lines of code (SLOC) by using Table 3 on the next page.

TABLE 3: UFP to SLOC Conversion

Language	SLOC/UFP
Ada	71
AI Shell	49
APL	32
Assembly	320
Assembly (Macro)	213
ANSI/Quick/Turbo Basic	64
Basic-Compiled	91
Basic-Interpreted	128
C	128
C++	29
ANSI COBOL 85	91
Fortran 77	105
Forth	64
Jovial	105
Lisp	64
Modula 2	80
Pascal	91

Cocomo 2.0 adjusts for the effects of reengineering in its effort estimate. When a project includes automatic translation, one needs to estimate:

- Automatic translation productivity (ATPROD), estimated from previous development efforts
- The size, in thousands of Source Lines of Code, of untranslated code (KSLOC) and of code to be translated (KASLOC) under this project.
- The percentage of components being developed from reengineered software (ADAPT)
- The percentage of components that are being automatically translated (AT).

Whereas in Cocomo 1.1, the effort equation is adjusted by 15 cost driver attributes, Cocomo 2.0 defines seven cost drivers (EM) for the Early Design estimate:

* Personnel capability
* Product reliability and complexity
* Required reuse
* Platform difficulty
* Personnel experience
* Facilities
* Schedule constraints.

Some of these effort multipliers are disaggregated into several multipliers in the Post-Architecture Cocomo 2.0 model.

Cocomo 1.1 models software projects as exhibiting decreasing returns to scale. Decreasing returns are reflected in the effort equation by an exponent for SLOC greater than unity. This exponent varies among the three Cocomo 1.1 development modes (organic, semidetached, and embedded). Cocomo 2.0 does not explicitly partition projects by development modes. Instead the power to which the size estimate is raised is determined by five scale factors:

* Precedentedness (how novel the project is for the organization)
* Development flexibility
* Architecture/risk resolution
* Team cohesion
* Organization process maturity.

Once an architecture has been established, the Cocomo 2.0 Post-Architecture model can be applied. Use of reengineered and automatically translated software is accounted for as in the Early Design equation (ASLOC, AT, ATPROD, and AAM). Breakage (BRAK), or the percentage of code thrown away due to requirements change is accounted for in 2.0. Reused software (RUF) is accounted for in the effort equation by adjusting the size by the adaptation adjustment multiplier (AAM). This multiplier is calculated from estimates of the percent of the design modified (DM), percent of the code modified (CM), integration effort modification (IM), software understanding (SU), and assessment and assimilation (AA). Seventeen effort multipliers are defined for the Post-Architecture model grouped into four categories:

* Product factors
* Platform factors
* Personnel factors
* Project factors.

These four categories parallel the four categories of Cocomo 1.1 - product attributes, computer attributes, personnel attributes and project attributes, respectively. Many of the seventeen factors of 2.0 are similar to the fifteen factors of 1.1. The new factors introduced in 2.0 include required reusability, platform experience, language and tool experience, personnel continuity and turnover, and a factor for multi-site development. Computer turnaround time, the use of modern programming practices, virtual machine experience, and programming language experience, which were effort multipliers in Cocomo 1.1, are removed in Cocomo 2.0.

A single development schedule estimate is defined for all three Cocomo 2.0 models, as shown in the second row of Table 2.

Putnam Software Equation

The Software Lifecycle Model (SLIM), available from Quantitative Software Management, is another popular empirically-based estimation model. An equation relating product size, effort, schedule, and productivity, known as the Software Equation, is a major part of SLIM. The fifth row of Table 1 and third row of 2 provide the effort and schedule equations, respectively, of the model. One interesting difference of the Software Equation over the other models is that the schedule equation needs to be solved before an effort estimate can be generated.

The Software Equation in Table 1 is written in two forms. The first equation is the complete equation as derived from the data collected by Larry Putnam and his associates at QSM. However a simpler form is also shown. This computational form of the effort equation can be used for medium and large projects where the effort is greater than or equal to 20 staff-months. It is derived for projects that achieve their minimum schedule. The minimum schedule length is constrained by an empirical relationship limiting how fast project staff can be built up.

The schedule equation in 2 for the minimum time can be applied if the computed minimum development time is greater than 6 calendar months. The computational form of the Software Equation is built on the premise that there is a minimum development time below which one cannot feasibly develop a given system. The general form of the Software Equation shows that the trade-off between effort and schedule is given by a fourth-power relationship. Hence an increase in the planned schedule can dramatically lower the resulting effort.

Application of the Software Equation requires prior determination of a productivity parameter, P. One should estimate the productivity parameter from historical data on similar systems within an organization using SLIM. If no historical data exists for the type of system one needs to develop, developing an accurate productivity parameter will be difficult. However, Putnam (1992) provides representative productivity parameters for systems of various types (e.g., real time systems, telecommunications systems, business systems). Putnam’s data on the productivity parameter in Table 2.3 is reproduced on the next page as Table 4.

Table 4: Putnam Productivity Parameter

Productivity Index	Productivity Parameter	Application Type
1	754	
2	987	Microcode
3	1,220	
4	1,597	Firmware (ROM)
5	1,974	Real-time embedded, Avionics
6	2,584	
7	3,194	Radar systems
8	4,181	Command and control
9	5,186	Process control
10	6,765	
11	8,362	Telecommuncations
12	10,946	
13	21,892	
14	13,530	Systems software, Scientific systems
15	17,711	
16	28,657	Business systems
17	35,422	
18	46,368	
19	57,314	
20	75,025	
21	92,736	
22	121,393	
23	150,050	
24	196,418	
25	242,786	Highest value found so far
26	317,811	
27	392,836	
28	514,229	
29	635,622	
30	832,040	
31	1,028,458	
32	1,346,269	
33	1,664,080	
34	2,178,309	
35	2,692,538	
36	3,524,578	

The Software Equation also requires an estimate of a special skills factor, B, which is a function of system size. Table 5 provides values of B.

Table 5: Putnam Special Skills Factor

Size (SLOC)	B
5-15K	0.16
20K	0.18
30K	0.28
40K	0.34
50K	0.37
>70K	0.39

An advantage of SLIM over Intermediate Cocomo 1.1 or Cocomo 2.0 is that there are far fewer parameters needed to generate an estimate. Also, as long as the type of system you want to estimate is similar in nature to systems developed before by your software organization, the historical data is probably readily available to compute a productivity parameter.

Putnam (1992) provides many useful and helpful guidelines in project management, often based on the distribution of labor over the lifecycle assumed in SLIM. For example, guidance is given on the time at which peak manpower should occur in the schedule, the number of people that should be working at the peak time, and average staffing over the schedule.

PRICE-S

Lockheed Martin Life Cycle Cost Estimating Systems' PRICE-S is a proprietary, empirically-based cost model. Since it is a proprietary model, complete information about the internals of the model are unavailable. Some details, however, are available about the model. Unlike other models, PRICE-S uses machine instructions, not source lines of code, as its main cost driver. You can hire consulting services from Lockheed Martin to exercise PRICE-S. However, if you need to use a consultant anyway to perform your estimates, the fact that PRICE-S is proprietary and requires a consultant to utilize may not be a problem to your organization. PRICE-S is one of the earliest and most successful models that have been developed.

Function Point Cost Estimation

All cost models that use source lines of code as their major cost driver can also use Function Points as their major cost driver. One can use Function Point estimation in a source lines of code-based model by using a lookup table to convert Function Point estimates to source lines of code. Recently there has been a great deal of interest in function points because, from an estimators viewpoint, function points can be easily assessed early in the products life cycle before any lines of code can be estimated and specific languages can be selected. Being able to estimate project costs and schedules directly from function points thus provides advantages to the estimator in being able to accurately estimate sooner. Several products and models are available to support function point estimation.

Capers Jones and his company, Software Productivity Research (SPR) provide two Microsoft Windows-based products, Checkpoint and Function Point Workbench, to aid in estimating and managing software projects. The product takes an artificial intelligence approach based on data collected by SPR on 5,200 projects. Details of the internals of the model and the estimating equations are proprietary, and thus no data is available for analysis here.

Other Function Point-based models have been identified by Matson (1994), but no commercial products based on these models are known. Table 1 shows equations for estimating effort for three Function Point-based models, the Albrecht and Gaffney Model, the Kemerer Model, and the Matson, Barnett, and Mellichamp Model. Any of these models can be easily implemented in a spread sheet.

Other Models

Matson (1994) lists other source lines of code-based effort estimation models, but no products implementing these models are commercially available. Table 1 shows effort equations for the Walston-Felix Model, the Bailey-Basili Model, and the Doty Model. Any of these models, too, can be easily implemented in a spread sheet.

Appendix A

Bibliography

Boehm, B., *Software Engineering Economics* Prentice-Hall, Inc., 1981.

Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., Selby, R., *Cost Models for Future Software Life Cycle Processes*, http://sunset.usc.edu/COCOMOII/Cocomo.html,1995.

Boehm, B., Clark, B., Horowitz, E., Westland, C., Madachy, R., Selby, R., *Cocomo 2.0 Model User's Guide*, http://sunset.usc.edu/COCOMOII/Cocomo.html, 1995.

Iuorno, R., Vienneau, R., *Software Measurement Models*, A DACS State of the Art Report, August 1987.

Martin, R., "Evaluation of Current Software Costing Tools," *ACM Sigsoft Notes*, vol. 13, no. 3, July 1988, pp. 49-51.

Matson, J., Barrett, B., Mellichamp, J., "Software Development Cost Estimation Using Function Points", *IEEE Transactions: Software Engineering*, vol. 20, no. 4, April 1994, pp. 275-287.

Putnam, L., Myers, W., *Measures for Excellence*, Yourdon Press, 1992.

Pressman, R., *Software Engineering: A Practitioner's Approach*, Fourth Edition, McGraw-Hill, 1997.