Estimating Software Costs
This article describes the cost estimation lifecycle and a process to estimate project volume.
Author: William Roetzheim
Co-Founder, Cost Xpert Group, Inc.

Estimating Software Costs

In extensive research of over 20,000 software development projects spanning 18 years, it is found that more projects are doomed from poor cost and schedule estimates than they ever are from technical, political, or development team problems. Yet, so few companies and individuals really understand that software estimating can be a *science*, not just an *art*. It really is possible to accurately and consistently estimate development costs and schedules for a wide range of projects. This series of 4 articles will provide you the tools you need to understand step-by-step approaches to estimating the cost and schedule for your projects. Although there is a wide range of software cost estimating tools on the market to help with this process, we will focus in this series of articles on understanding the fundamental concepts. You will be able to implement the concepts in these articles using nothing more complicated than a spreadsheet.

We'll start with this article by covering the various methods of estimating the size of a program (called *program volume*). We'll discuss the traditional measures of Lines of Code and Function Points, plus introduce other approaches. At the end of this article, we'll show you how to prepare a preliminary, unadjusted estimate using this information.

Article 2, *Project Cost Adjustments,* covers the concept of project cost adjustments for variations in the project environment. At the end of this article, you will be able to create an accurate estimate of the time and cost required to develop a new application.

Article 3, *Dealing with Reuse,* explains how to quantify the impact of software reuse and commercial components/libraries on your estimate.

Finally, Article 4, *Creating the Project Plan* describes how to use your insight into project cost and schedule to create a complete project plan.

The Estimating Lifecycle

First, it is important to recognize the limitations of software cost estimating at the macro level. As shown in the following figure, the typical accuracy of cost estimates varies based on the current software development stage. Early uncertainty in the estimate is largely based on variances in the input parameters

Cost Xpert Group, Inc.
www.costxpert.com
2990 Jamacha Rd, Suite 250
El Cajon, California 92019
Phone: 619.670.6168
Fax: 619.670.3230

to the estimate. Later uncertainty in the estimate is based on the variances of the estimating models.

Initially, at the concept stage, you may be presented with a vague definition of the project. Though the requirements may not yet be fully understood, the general purpose of the new software can be recognized. At this point, estimates with an accuracy of plus or minus fifty percent are typical for an experienced estimator using informal techniques (i.e., historical comparisons, group consensus, and so on).

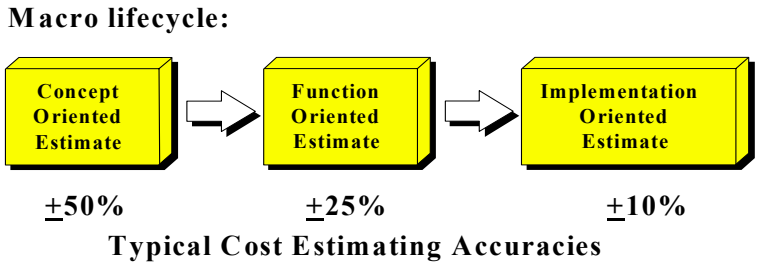

FIGURE 1: MACRO LIFECYCLE

After the requirements are reasonably well understood, a function-oriented estimate may be prepared. At that point, estimates with an accuracy of plus or minus twenty-five percent are typical for an experienced estimator, using the techniques described above.

Finally, after the detailed design is complete, an implementation-oriented estimate may be prepared. This estimate is typically accurate within plus or minus ten percent.

The key is to recognize the importance of periodic re-estimates throughout the project lifecycle, thereby identifying problems early enough to take corrective action.

Estimating Program Volume

The first step in preparing an estimate is to come up with an estimate of the project volume. One measure of program volume is the number of Source Lines of Code, or SLOC. A SLOC is a human written line of code that is not a blank line or comment. Do not count the same line more than one time even if the code is included multiple times in an application. We typically work with a related number, thousands of SLOC, or KSLOC, when estimating. The Constructive Cost Model, or COCOMO popularized SLOC as an estimating metric. The basic COCOMO model and the new COCOMO II model remain the most common estimating approaches.

Cost Xpert Group, Inc.
www.costxpert.com
2990 Jamacha Rd, Suite 250
El Cajon, California 92019
Phone: 619.670.6168
Fax: 619.670.3230

Let's jump ahead and look at how we can convert from the number of KSLOC to an estimate for the project. We will then discuss approaches to estimating KSLOC in more detail.

Let's begin with the simplest estimate. If you are aware of the number of KSLOC your developers must write, and you know the effort required per KSLOC, then you could multiply these two numbers together to arrive at the person months of effort required for your project. This concept is the heart of the estimating models. The following table shows some common values that researchers have found for this linear productivity factor.

Project Type	Linear Productivity Factor
COCOMO II Default	3.13
Embedded development	3.60
E-Commerce development	3.08
Web development	2.51
Military development	3.97

OK, let's apply this approach. Suppose we were going to build an e-commerce system consisting of 15,000 lines of code. How many person months of effort would this take using just this equation?

The answer is computed as follows:

Productivity * KSLOC = 3.08 * 15 = Effort = 46 Person-Months

If all of your projects are small, then you can use this basic equation. Researchers have found, however, that productivity does vary with project size. In fact, large projects are significantly less productive than small projects. The probable causes are a combination of increased coordination and communication time, plus more rework required due to misunderstandings.

This productivity decrease with increasing project size is factored in by raising the number of KSLOC to a number greater than 1.0. This exponential factor then penalizes large projects for decreased efficiency. The following table shows some typical size penalty factors for various project types.

Project Type	Exponential size penalty factor
COCOMO II Default	1.072
Embedded development	1.111
E-Commerce development	1.030
Web development	1.030
Military development	1.072

Cost Xpert Group, Inc.
www.costxpert.com
2990 Jamacha Rd, Suite 250
El Cajon, California 92019
Phone: 619.670.6168
Fax: 619.670.3230

So, after we do a size penalty adjustment, how many person months of effort would our 15,000 lines of code e-commerce system require? The answer is computed as follows:

$$\text{Productivity} * \text{KSLOC}^{\text{Penalty}} = 3.08 * 15^{\,1.030}$$
$$= 3.08 * 16.27 = \text{Effort}$$
$$= 50 \text{ Person-Months}$$

OK, all of this is pretty straightforward. The next logical question is, "How do I know my project will end up as 15,000 SLOC?"

There are two approaches to answering this question I will address in this article, direct estimation and function points with backfiring. Using either approach, the fundamental input variables are determined through expert opinion, often with your developers as the experts. The Delphi technique is a good way to crosscheck the input variables.

Normally, the first step in estimating the number of lines of code is to break the project down into modules or some other logical grouping. For example, a very high level breakdown might be front-end processes; middle-tier processes; and database code. Your developers then use their experience building similar systems to estimate the number of lines of code required.

We strongly recommend that you obtain three estimates for each input variable: A best-case estimate; a worst-case estimate; and an expected case estimate. With these three inputs, we can then calculate the mean and standard deviation as follows:

$$Mean = \frac{(best + worst + (4 \times expected))}{6}$$

EQUATION 1: CALCULATING THE MEAN

$$Standard\ \ Deviation = \frac{(worst - best)}{6}$$

EQUATION 2: CALCULATING THE STANDARD DEVIATION

The standard deviation is a measure of how much deviation can be expected in the final number. For example, the mean plus three times the standard deviation will ensure that there is a 99% probability that your project will come in *under* your estimate.
For more information, refer to the outside reference *Software Engineering and Project Management,* Boehm, B., IEEE Press, 1987.

Cost Xpert Group, Inc.
www.costxpert.com
2990 Jamacha Rd, Suite 250
El Cajon, California 92019
Phone: 619.670.6168
Fax: 619.670.3230

Estimating Function Points

An alternative to direct SLOC estimating is to start with function points, then use a process called backfiring to convert from function points to SLOC. Function Points were first utilized by IBM as a measure of program volume. The idea is simple. The program's delivered functionality (and hence, cost) is measured by the number of ways it must interact with the users.

To determine the number of Function Points, start by estimating the number of external inputs, external interface files, external outputs, external queries, and logical internal tables.

External inputs are largely your data entry screens. If a screen contains a tabbed notebook or similar metaphor then each tab would count as a separate external input. External interface files are file based inputs or outputs. Each record format within the file, or in the case of XML, each data object type, would count as a separate interface file even if residing in the same physical file. External outputs are your reports. External queries are message or external function based communication into or out of your application. Finally, logical internal tables are the number of tables in the database assuming the database was 3rd normal form or better.

To convert from these raw values into an actual count of function points, you multiply the raw numbers by a conversion factor from the following table:

Raw Type	Function Point Conversion Factor
External inputs	4
External interface files	7
External outputs	5
External queries	4
Logical internal tables	10

So, if we had a system consisting of 25 data entry screens, 5 interface files, 15 reports, 10 external queries, and 20 logical internal tables, how many function points would we have?

The answer is computed as follows:

$$(25*4)+(7*5)+(15*5)+(10*4)+(20*10)=450 FunctionPoints$$

The only remaining step is to use backfiring to convert from function points to an equivalent number of SLOC. This can be done using a table of language equivalencies. Some common values are as follows:

Cost Xpert Group, Inc.
www.costxpert.com
2990 Jamacha Rd, Suite 250
El Cajon, California 92019
Phone: 619.670.6168
Fax: 619.670.3230

Language	SLOC per Function Point
C++ default	53
Cobol default	107
Delphi 5	18
HTML 4	14
Java 2 default	46
Visual Basic 6	24
SQL default	13

So, to implement the above project (450 function points) using Java 2 would require approximately the following number of SLOC:

$$450 * 46 = 20{,}700 SLOC$$

And would require the following effort to implement, assuming that this was an e-commerce system:

$$\begin{aligned} \text{Productivity} * \text{KSLOC}^{\text{Penalty}} &= 3.08 * 20.7^{\,1.030} \\ &= 3.08 * 22.67 = \text{Effort} \\ &= 70 \text{ Person-Months} \end{aligned}$$

As discussed in our article, there are also other approaches to calculating equivalent SLOC from a higher-level input value. These other approaches include Internet points, Domino points, and class-method points to name just a few. All of them work in a fashion analogous to function points as just described.

In our next installment, *Project Cost Adjustments,* we will cover the concept of project cost adjustments for variations in the project environment.

Cost Xpert Group, Inc.
www.costxpert.com
2990 Jamacha Rd, Suite 250
El Cajon, California 92019
Phone: 619.670.6168
Fax: 619.670.3230