Estimating and Managing Project Scope for New Development

Many consider estimating project scope to be the most difficult part of software estimation. Parametric models have been shown to give accurate estimates of cost and duration when given accurate inputs of the project scope, but how do you input scope early in the life cycle when the requirements are still vaguely understood? How can scope be estimated, quantified, and documented in a manner that is understandable to management, end users, and estimating tools? This article focuses on scope estimates for new development, and is applicable for the new development portion of maintenance builds.

The Estimating Life Cycle
First, it is important to recognize the limitations of software cost estimating at the macro level. As shown in Figure 1, 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 estimate’s input parameters. Later uncertainty in the estimate is based on the variances of the estimating models.

The percentages shown in Figure 1 match this author’s personal experience and are roughly comparable with figures found in the Project Management Institute’s “A Guide to the Project Management Body of Knowledge” [1]. However, actual numbers will vary widely based on the type of applications involved, the estimators’ experience and policies, and other factors.

Initially, at the concept stage you may be presented with a vague project definition. 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 ± 50 percent are typical for an experienced estimator using informal techniques (i.e., historical comparisons, group consensus, and so on).

After the requirements are reasonably well understood, a function-oriented estimate may be prepared. At this point, estimates with an accuracy of ± 25 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 ± 10 percent.

Figure 1: Macro Life Cycle

Macro Life Cycles

<table>
<thead>
<tr>
<th>Concept-Oriented Estimate</th>
<th>Function-Oriented Estimate</th>
<th>Implementation-Oriented Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>±50%</td>
<td>±25%</td>
<td>±10%</td>
</tr>
</tbody>
</table>

Typical Cost Estimating Accuracies

“Estimating Program Scope
The first step in preparing an estimate is to determine an estimate of the project scope, or volume. Scope is typically estimated using a variety of metrics, as different portions of the application may be compatible with different scope metrics.

One measure of program scope is the number of source lines of code (SLOC). A source line of code is a human-written line of code that is not a blank line or comment. Do not count the same line more than once even if the code is included multiple times in an application. We typically work with a related number – thousands of SLOC (KSLOC) – when estimating. The Constructive Cost Model (COCOMO) popularized SLOC as an estimating metric. The basic COCOMO model and the new COCOMO II model remain the most well-known estimating approaches because of their prevalence in both academic research settings and as models embedded into estimating tools.

Let us 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.

Begin with the simplest estimate as shown in Table 1. If you are aware of the number of KSLOC your developers must write, and you know the effort required per KSLOC, you then 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. Table 1 shows some common values that Cost Xpert researchers have found for this linear productivity factor. The COCOMO II value comes from research by Barry Boehm [2] at the University of Southern California. The values for embedded, e-commerce, and Web development come from the Cost Xpert Group’s [3] research working with a vari-
Estimating and Managing Project Scope for New Development

Estimate Example

Linear Productivity Factor

<table>
<thead>
<tr>
<th>Project Type</th>
<th>Linear Productivity Factor (Person Months/KSLOC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COCOMO II Default</td>
<td>3.13</td>
</tr>
<tr>
<td>Embedded Development</td>
<td>3.60</td>
</tr>
<tr>
<td>E-Commerce Development</td>
<td>3.08</td>
</tr>
<tr>
<td>Web Development</td>
<td>2.51</td>
</tr>
</tbody>
</table>

Table 1: Estimate Example

Exponential Size Penalty Factor

<table>
<thead>
<tr>
<th>Project Type</th>
<th>Exponential Size Penalty Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>COCOMO II Default</td>
<td>1.072</td>
</tr>
<tr>
<td>Embedded Development</td>
<td>1.111</td>
</tr>
<tr>
<td>E-Commerce Development</td>
<td>1.030</td>
</tr>
<tr>
<td>Web Development</td>
<td>1.030</td>
</tr>
</tbody>
</table>

Table 2: Typical Size Penalty Factors

The productivity decrease with increasing project size is factored in by raising the number of KSLOC to a power greater than 1.0. This exponential factor then penalizes large projects for decreased efficiency. Table 2 shows some typical size penalty factors for various project types. Again, the COCOMO II value comes from work by Boehm [2], and values for embedded, e-commerce, and Web development come from work by Cost Xpert Group [3] and its customers. These values have been validated by hundreds of Cost Xpert Group customers/projects, and are updated over time as warranted by the research. Note that because the size factor is an exponential factor rather than linear, it does not change with project size, but changes in impact on the end result with project size.

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:

Effort = Productivity x KSLOCPenalty = 3.08 x 15 = 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 power greater than 1.0. This exponential factor then penalizes large projects for decreased efficiency. Table 2 shows some typical size penalty factors for various project types. Again, the COCOMO II value comes from work by Boehm [2], and values for embedded, e-commerce, and Web development come from work by Cost Xpert Group [3] and its customers. These values have been validated by hundreds of Cost Xpert Group customers/projects, and are updated over time as warranted by the research. Note that because the size factor is an exponential factor rather than linear, it does not change with project size, but changes in impact on the end result with project size.

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:

Effort = Productivity x KSLOCPenalty = 3.08 x 15 = 46 Person Months

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 that I will address: direct estimation and function points (FPs) with backfiring. Using either approach, the fundamental input variables are determined through expert opinion, often with your developers as the experts. The Delphi technique, involving multiple experts iterating toward a consensus decision, is a good way to cross-check the input variables.

Normally, the first step in estimating the number of LOC is to break down the project 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 LOC 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, you can then calculate the mean and standard deviation as follows:

\[
\text{Mean} = \frac{\text{best} + \text{worst} + (4 \times \text{expected})}{6}
\]

\[
\text{Standard Deviation} = \frac{\text{worst} - \text{best}}{6}
\]

The standard deviation is a measure of how much deviation can be expected in the final number. For example, if the statistical description of the project is correct and we ignore risk factors not included in the statistical spread, the mean plus three times the standard deviation will ensure that there is a 99 percent probability that your project will come in under your estimate.

For more information, refer to [4].

Estimating Function Points

An alternative to direct SLOC estimating is to start with FPs, then use a process called backfiring to convert from FPs to SLOC. Backfiring is described on page 6, and consists of converting from FPs to SLOC using a language-driven table lookup function. FPs were first utilized by IBM as a measure of program volume. Counting FPs has evolved over time as computer programming techniques and user interface metaphors became more complex; correct function point counting is defined in [5] and is often accomplished using certified FP counting specialists.

The original basic idea is simple and illustrates how it works at a simplified level. True FP counts are more complicated, of course. 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 FPs, start by estimating the number of external inputs, external interface files, external outputs, external queries, and logical internal files. External inputs are largely your data-entry screens. External interface files are file-based inputs or outputs. External outputs are your reports and static outputs. External queries are message or external function-based communication into or out of your application. Finally, logical internal files are the number of tables in the database, assuming the database was third normal form or better. As mentioned earlier, these definitions are simplified, but they serve to illustrate the basic concept.

To convert from these raw values into an actual count of FPs, you multiply the raw numbers by a conversion factor from Table 3 on page 6 (again, this approach is a simplification).
You might find that a new proposed project is much more complicated than your database of historic projects. Perhaps you can combine multiple historic projects, each corresponding to a piece of the new project, to arrive at a total estimate of the scope.

Note that it is better to use this approach to estimate scope and then use an estimating tool to estimate effort, rather than using this approach directly to estimate effort. Basically, the scope will be somewhat consistent between similar projects; however, the effort will have a high degree of variability due to things like the people doing the work, the standards and life cycles used, and the development environments. By using historic data to approximate scope and then using project-specific data for all of these other variables, you obtain a much more accurate effort estimate.

**Design to Budget and Time-Box Approaches**

It is not unusual for a software development budget to be defined before the requirements are defined or perhaps even understood. Market factors might drive the budget. Competitors might define the budget. Resource limitations might determine the budget. In these cases, does estimating make any sense? In fact, estimates are particularly critical under these circumstances.

The approach is to initialize an estimating tool with appropriate values for all of the environmental variables (e.g., development team capabilities, development language, life cycle, standard, etc.). Then, start plugging in values for scope until you obtain a scope estimate that meets the external budget constraint. This then becomes the amount of functionality that you can deliver for the specified budget.

Throughout the development process you must manage expectations to ensure that each step in the process is defining a system that is no larger in size than the budgeted scope. The requirements must be managed along with the design effort, the physical implementation, and so on.

**Project Type Taxonomies**

It is possible to use project type taxonomies to approximate the FP count of a system to be built (this approach was initially proposed by Capers Jones in “Estimating Software Costs” [6]). The values shown in Table 5 come from Cost Xpert Group research and vary somewhat from the specifics in [6]. It works as follows: In Table 5, select the numerical value that corresponds to your selected project.
scope. In other words, are you simply developing a function? Are you writing an object? Are you writing a library of objects? Is this a new shrink-wrap application, or a completely new system (e.g., missile system)?

Using Table 6, select the numerical value that corresponds to your selected project class. In other words, is this development for your personal use? Is it shareware? Is this a civilian-contract programming project? Is this a military project?

Using Table 7, select the numerical value that corresponds to your selected project type. In other words, is this a drag-and-drop fourth generation language development? Is this a batch program? Is it a client-server application? Is it a mathematical application? Is it a new social services program?

Add the three values just obtained together, and then raise this number to the 2.35 power as shown in the following equation:

$$\text{FPs} = \left(\text{Value}_{\text{Scope}} + \text{Value}_{\text{Class}} + \text{Value}_{\text{Type}}\right)^{2.35}$$

This will give you an approximate value for the number of FPs in the final delivered application. The actual values (e.g., 2.35) are simply mathematical curve-fitting techniques to force this early estimation equation to fit databases of historic projects.

Let us look at a few examples. Suppose we were asked by a commercial communication company to estimate the effort required to create an object that would perform some signal processing functions. This object will be our deliverable. We would use the following values:

**Scope** = Object
**Class** = Contract Project - Civilian
**Type** = Communications

$$\text{FPs} = \left(\text{Value}_{\text{Scope}} + \text{Value}_{\text{Class}} + \text{Value}_{\text{Type}}\right)^{2.35} = (2 + 7 + 11)^{2.35} = 1,141 \text{ FPs}$$

Now, suppose we were asked to create a user interface proof-of-concept for a fixed-asset tracking system for internal use only:

**Scope** = Proof of Concept
**Class** = Single Location-Internal
**Type** = No Programming (4GL/Drag and Drop)

$$\text{FPs} = \left(\text{Value}_{\text{Scope}} + \text{Value}_{\text{Class}} + \text{Value}_{\text{Type}}\right)^{2.35} = (5 + 5 + 1)^{2.35} = 280 \text{ FPs}$$

Finally, suppose we need to estimate the effort required to build a new welfare system to be used by a single state with consolidated rules (e.g., there would be no requirement to deliver tailored versions for different counties within the state):

**Scope** = External Application
**Class** = State Government-Federally Funded
**Type** = Social Services

$$\text{FPs} = \left(\text{Value}_{\text{Scope}} + \text{Value}_{\text{Class}} + \text{Value}_{\text{Type}}\right)^{2.35} = (9 + 13 + 15)^{2.35} = 4,845 \text{ FPs}$$

### Conclusion

While determining the scope of new development is never easy, there are techniques that should help you get into the right ballpark. Once there, it becomes a matter of tracking and managing to that scope, either by ensuring that requirements do not grow to exceed the budgeted scope or by using engineering change proposals to obtain additional resources and time when the requirements do exceed the planned scope.

### References


### About the Author

William Roetzheim has 25 years experience in the software industry and is the author of 15 software-related books and over 100 technical articles. He is the founder of the Cost Xpert Group, Inc., a Jamul-based organization specializing in software cost-estimation tools, training, processes, and consulting.

Cost Xpert Group
2990 Jamacha RD STE 250
Rancho San Diego, CA 92019
E-mail: william@costxpert.com