Estimating and Managing Project Scope for New Development

William Roetzheim
Cost Xpert Group

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 life cycle of software cost estimation is made of many parts, beginning with input parameters at the concept stage and continuing through the function and implementation stages. Many consider estimating project scope to be the most difficult part of software estimation. After all, how do you input scope early in the life cycle when the requirements are still vaguely understood? Consider also that scope must be estimated, quantified, and documented in a manner that is understandable to management, end users, and estimating tools. The focus in this article is scope estimates for new development, including 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.

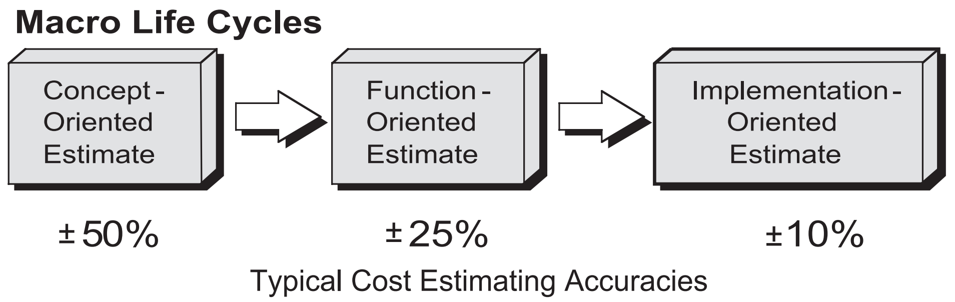
Initially, at the concept stage you may be presented with a vague project definition. Though the requirements may not yet be fully understood, the general purpose of the new software can be recognized. At this point, estimates with an accuracy of ± 50 percent are typical for an experienced estimator using informal techniques (i.e., historical comparisons, group consensus, and so on).

"The first step in preparing an estimate is to determine an estimate of the project scope, or volume."

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
Concept-Oriented Estimate
Function-Oriented Estimate
Implementation-Oriented Estimate
±50%
±25%
±10%
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 one time even if the code is included multiple times in an application[1]. 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 variety of organizations, including IBM and Marotz.

Now, let us apply this approach. Suppose we were going to build an e-commerce system consisting of 15,000 LOC. How many person-months of effort would this take using just this equation? The answer is computed as follows:

$$\text{Effort} = \text{Productivity} \times \text{KSLOC} = 3.08 \times 15 = 46 \text{ Person Months}$$

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:

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

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.

Project Type	Linear Productivity Factor (Person Months/KSLOC)
COCOMO II Default	3.13
Embedded Development	3.60
E-Commerce Development	3.08
Web Development	2.51

Table 1: *Estimate Example*

Project Type	Exponential Size Penalty Factor
COCOMO II Default	1.072
Embedded Development	1.111
E-Commerce Development	1.030
Web Development	1.030

Table 2: *Typical Size Penalty Factors*

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).

Raw Type	Function Point Conversion Factor
External Inputs	4
External Interface files	7
External Outputs	5
External Queries	4
Logical Internal Tables	10

Table 3: *Function Point Conversion Factor*

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

Table 4: *Language Equivalencies*

So, if we had a system consisting of 25 data-entry screens, five interface files, 15 reports, 10 external queries, and 20 logical internal tables, how many FPs would we have? The answer is computed as follows:

(25 x 4) + (5 x 7) + (15 x 5) + (10 x 4) + (20 x 10) = 450 FPs

Backfiring

The only remaining step is to use backfiring to convert from FPs to an equivalent number of SLOC. This is done using a table of language equivalencies. Some common values are shown in Table 4 (C++, COBOL, and SQL from work by Capers Jones [6] and other values from research by Cost Xpert Group [3]):

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

450 x 46 = 20,700 SLOC

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

$$\text{Effort} = \text{Productivity} \times \text{KSLOC}^{\text{Penalty}} = 3.08 \times 20.7^{1.030} = 3.08 \times 22.67 = 69.8 \text{ Person Months}$$

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 FPs as just described.

Table 5: *Project Scope Table*

Function	1
Object	2
Object Library	4
Proof of Concept	5
Evolutionary Prototype	6
Internal Application	8
External Application	9
Shrink-Wrap Application	10
Component of System	11
New System	12
Compound System	13

Heuristic Approaches to Approximating Scope

Estimating Scope by Analogy

This is the software equivalent of market comps in appraising real estate: You look for a project that is as close as possible to your project. Count the physical LOC or function points in that application. Then, use a detailed analysis to adjust things up or down based on differences between the proposed project and this historic project.

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:

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

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

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

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)

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

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

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

Individual Use	1
Shareware	2
Academic/Engineering	3
Single Location - Internal	5
Multilocation - Internal	6
Contract Project - Civilian	7
Contract Project - Local Government	8
Marketed Commercially	9
State Government	11
State Government - Federally Funded	13
Federal Project	14
Military Project	15

Table 6: *Project Class Table*

No Programming (4GL/Drag and Drop)	1
Batch	2
3GL Programming	4
Embedded - Single Board	5
Database Oriented	6
Client-Server	8
Mathematical	9
Systems	10
Communications	11
Process Control	12
Embedded - Multi-Board	13
Embedded - Complete System	14
Social Services	15

Table 7: *Project Type Table*

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

1. Project Management Institute. A Guide to the Project Management Body of Knowledge (PMBOK Guide). 3rd ed. Newton Square, PA: PMI, 2004.
2. Boehm, Barry, et al. Software Cost Estimation With COCOMO II. Upper Saddle River, N.J.: Prentice-Hall, 2000.
3. Cost Xpert Group. "Data Load 3.3(b)." Internal Report. Rancho San Diego, CA: Cost Xpert Group, 2004.
4. Boehm, Barry. Software Engineering and Project Management. IEEE Press, 1987.
5. ISO [International Organization for Standardization]/International Electrotechnical Commission 20926: 2003 <www.iso.org>.
6. Jones, Capers. Estimating Software Costs. McGraw-Hill, New York, 1998 <www.iso.org/en/prods-services/IOSstore/store.html>.

Note

1. This is a slightly simplified version of the definition from the Software Engineering Institute's Definition Checklist for a Logical Source Statement by R. Park in "Software Size Measurement: A Framework for Counting Source Statements" SEI, Pittsburgh, PA, 1992.

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