Software Estimation

Accurately estimating software size, cost, effort, and schedule is probably the biggest challenge facing software developers today.

A discussion of metrics and metrics programs cannot be complete without a discussion of software estimation. Unfortunately, software estimation is an extensive subject, and a complete discussion of the topic would be inappropriate here. Instead, a quick overview follows.

Estimating Software Size

An accurate estimate of software size is an essential element in the calculation of estimated project costs and schedules. The fact that these estimates are required very early on in the project (often while a contract bid is being prepared) makes size estimation a formidable task.

Initial size estimates are typically based on the known system requirements. You must hunt for every known detail of the proposed system, and use these details to develop and validate the software size estimates.

In general, you present size estimates as lines of code (KSLOC or SLOC) or as function points. There are constants that you can apply to convert function points to lines of code for specific languages, but not vice versa. If possible, choose and adhere to one unit of measurement, since conversion simply introduces a new margin of error into the final estimate.

Regardless of the unit chosen, you should store the estimates in the metrics database. You will use these estimates to determine progress and to estimate future projects. As the project progresses, revise them so that cost and schedule estimates remain accurate.

The following section describes techniques for estimating software size.

Developer Opinion

Developer opinion is otherwise known as guessing. If you are an experienced developer, you can likely make good estimates due to familiarity with the type of software being developed.

Previous Project Experience

Looking at previous project experience serves as a more educated guess. By using the data stored in the metrics database for similar projects, you can more accurately predict the size of the new project.

If possible, the system is broken into components, and estimated independently.

Count Function Blocks

The technique of counting function blocks relies on the fact that most software systems decompose into roughly the same number of "levels". Using the information obtained about the proposed system, follow these steps:

1. Decompose the system until the major functional components have been identified (call this a function block, or software component).
2. Multiply the number of function blocks by the expected size of a function block to get a size estimate.
3. Decompose each function block into subfunctions.
4. Multiply the number of subfunctions by the expected size of a subfunction to get a second size estimate.
5. Compare the two size estimates for consistency.

Compute the expected size of a function block and/or a subfunction with data from previous projects that use similar technologies and are of similar scope.

If there are no previous developments on which to base expected sizes, use the values 41.6 KSLOC and 4.16 KSLOC for the expected size of function blocks and subfunctions respectively. These values were presented by Britchner and Gaffney (1985) as reasonable sizes for aerospace projects (real-time command and control systems).

Function Point Analysis

Function points allow the measurement of software size in standard units, independent of the underlying language in which the software is developed. Instead of counting the lines of code that make up a system, count the number of externals (inputs, outputs, inquiries, and interfaces) that make up the system.

There are five types of externals to count:

1. external inputs - data or control inputs (input files, tables, forms, screens, messages, etc.) to the system
2. external outputs - data or control outputs from the system
3. external inquiries - I/O queries which require a response (prompts, interrupts, calls, etc.)
4. external interfaces - libraries or programs which are passed into and out of the system (I/O routines, sorting procedures, math libraries, run-time libraries, etc.)
5. internal data files - groupings of data stored internally in the system (entities, internal control files, directories)

Apply these steps to calculate the size of a project:

1. Count or estimate all the occurrences of each type of external.
2. Assign each occurrence a complexity weight.
3. Multiply each occurrence by its complexity weight, and total the results to obtain a function count. Complexity weights are listed in Figure C.1.

Figure C.1 : Complexity weights.

	Complexity		
Description	**Low**	**Medium**	**High**
External inputs	3	4	6
External outputs	4	5	7
External inquiries	3	4	6
External interfaces	5	7	10
Internal files	7	10	15

4. Multiply the function count by a value adjustment multiplier (VAM) to obtain the function point count.

The VAM is calculated as follows:

$$VAM = \sum_{i=1}^{14} Vi \bullet 0.01 + 0.65$$

where V_i is a rating of 0 to 5 for each of the following fourteen factors ($_i$). The rating reflects how each factor affects the software size.

1. data communications
2. distributed functions
3. performance
4. heavily used operational configuration
5. transaction rate
6. on-line data entry
7. design for end user efficiency
8. on-line update of logical internal files
9. complex processing
10. reusability of system code
11. installation ease
12. operational ease
13. multiple sites
14. ease of change

Assign the rating of 0 to 5 according to these values:

0 - factor not present or has no influence
1 - insignificant influence
2 - moderate influence
3 - average influence
4 - significant influence
5 - strong influence

Function point analysis is extremely useful for the transaction processing systems that make up the majority of MIS projects. However, it does not provide an accurate estimate when dealing with command and control software, switching software, systems software or embedded systems.

To learn more about function point analysis, consider joining the International Function Point Users Group (IFPUG). This group has developed the *Function Point Counting Practices Manual*, which describes how to apply function points to software development.

Count Externals

Counting externals is the application of function point analysis (presented in the previous section) to real-time embedded systems. Instead of a function point count, the end result is an estimated size in KSLOC. This is based on estimated counts of the following externals (defined in the previous section):

- external inputs
- external outputs
- external inquiries
- external interfaces

The procedure for estimating the size of a new project in KSLOC is as follows:

1. Identify the number of program externals for each major program component (function block or software component). Either estimate all four externals, or work with these three externals: inputs, outputs, and inquiries.
2. Apply one of the following formulas:

   KSLOC = 12.288 + 0.030E (where E is the sum of all four externals)

   KSLOC = 13.94 + 0.034A (where A is the sum of the three externals)

If the KSLOC and externals information is stored in a metrics database from previous projects, you can develop specific formulas to more accurately reflect an organization's software development potential. Do this by plotting size (y-axis) versus counts of externals (x-axis), and fitting a line that best represents the data. The formula that describes this line is the estimating formula.

Combining Estimates

A good method for improving the accuracy of estimates is to estimate several ways, and then calculate a weighted average of the estimates. For example, to average four estimates obtained by different techniques:

1. Assign each estimate a weight (such that all the weights sum to 1.0). A small weight indicates confidence in the estimate while a large weight indicates uncertainty in the estimate.
2. Compute the weighted size for each estimate (size • weight).
3. Sum the weighted sizes to get the combined estimate.

Figure C.2 illustrates the calculation of a combined size estimate from four existing estimates for a result of 129 KSLOC.

Figure C.2 : Calculating a combined size estimate.

KSLOC	Weight	Weighted Size
120	0.30	36
160	0.40	64
100	0.20	20
90	0.10	9
		129

Recommendations for Estimating Size

Estimate the software size using a number of techniques, and then average these results to produce a combined estimate. As the metrics program matures, use the data collected from previous projects to develop specific estimating procedures and formulas.

Remember to re-estimate as the project progresses. Software estimates usually increase over the life of the project, and you should adjust cost and effort estimates accordingly.

Estimate the size of each program component (function block or software component) independently and relate this size to similar products and project components.

Estimating Software Cost

The cost of medium and large software projects is determined by the cost of developing the software, plus the cost of equipment and supplies. The latter is generally a constant for most projects.

The cost of developing the software is simply the estimated effort, multiplied by presumably fixed labour costs. For this reason, we will concentrate on estimating the development effort, and leave the task of converting the effort to dollars to each company.

Estimating Effort

There are two basic models for estimating software development effort (or cost): holistic and activity-based. The single biggest cost driver in either model is the estimated project size.

Holistic models are useful for organizations that are new to software development, or that do not have baseline data available from previous projects to determine labour rates for the various development activities.

Estimates produced with activity-based models are more likely to be accurate, as they are based on the software development rates common to each organization. Unfortunately, you require related data from previous projects to apply these techniques.

Holistic Models for Cost Estimating

Holistic models relate size, effort, and (sometimes) schedule by applying equations to determine the overall cost, and then applying a percent of the overall cost to each development activity. They do not consider the actual labour rates and costs of each activity.

Popular holistic models include the following:

- SDM (Software Development Model - Putnam - 1978)
- SLIM (Software Lifecycle Management - Putnam - 1979)
- COCOMO (Constructive Cost Model - Boehm - 1981)
- COPMO (Cooperative Programming Model - Conte, Dunsmuir, Shen - 1986)

Of these models, COCOMO is most widely used, and will suffice if there is insufficient data to carry out activity-based cost estimation.

Basic COCOMO

COCOMO comes in three levels (basic, intermediate, and detailed) with each providing progressively more accurate estimates. This section gives a brief overview of basic COCOMO.

Basic COCOMO is provided for three operational modes: organic, semi-detached, and embedded.

You would apply the organic mode to projects that have a small, experienced development team which is developing familiar applications in a familiar environment.

You apply the embedded mode to large projects, especially when the project is unfamiliar or there are severe time constraints.

The semi-detached mode is for projects somewhere in between.

Figure C.3 : Equations for basic COCOMO.

Mode	**Effort (Cost)**	**Schedule**
Organic	$\text{Effort} = 2.4(\text{Size})^{1.05}$	$\text{Time} = 2.5(\text{Effort})^{0.38}$
Semi-detached	$\text{Effort} = 3.0(\text{Size})^{1.12}$	$\text{Time} = 2.5(\text{Effort})^{0.35}$
Embedded	$\text{Effort} = 3.6(\text{Size})^{1.20}$	$\text{Time} = 2.5(\text{Effort})^{0.32}$

Effort is presented in person months, size is estimated in KSLOC, and time is estimated in months.

For more details on the basic COCOMO model, and the intermediate and detailed models, refer to Barry Boehm's book, *Software Engineering Economics*.

Activity-Based Models for Cost Estimating

The activity-based model uses data from the metrics database to determine the labour rates for the various development activities. For this reason, you can only apply it once the metrics program is established and there is a baseline from which to work.

In this document, the labour rate is defined as PH/SLOC (PH is person hours). Using this definition, the formula for estimating the cost of a project is as follows:

$$\text{Effort} = \sum_{i=1}^{n} (\text{PH} / \text{SLOC})_{i,\text{new}} \bullet \text{SLOC}_{\text{new}} + \sum_{i=1}^{n} (\text{PH} / \text{SLOC})_{i,\text{reused}} \bullet \text{SLOC}_{\text{reused}}$$

where $(\text{PH} / \text{SLOC})_{i,j}$ is the labour rate for activity i and code class j (j equals new or reused), and $(\text{SLOC})_j$ is the estimated size (in SLOC) of code for class j. The resulting effort is given in person hours.

Estimating Software Schedule

There are many tools on the market (such as Timeline, MacProject, OnTarget, etc.) which help develop Gantt and PERT charts to schedule and track projects.

These programs are most effective when you break the project down into a Work Breakdown Structure (WBS), and assign estimates of effort and staff to each task in the WBS.