A Method for Improving Developers' Software Size Estimates

Lawrence H. Putnam, Douglas T. Putnam, and Donald M. Beckett
Quantitative Software Management, Inc.

Traditional software estimating is effort-based and follows a bottom-up approach. This approach does not show the impact of different team sizes or the impact of schedule, cost, and quality constraints. The authors propose a method that decomposes programming artifacts into elementary units of work that form the size used for model-based estimating. The process is simple to implement, flexible, can be tuned with actual project performance data, and fosters developer buy-in by involving them in the estimating process.

It is common to hear this question during project development: "How large is this project?"

One type of answer might be, "Oh, let me see. I believe that this is about a 500 effort-hour project."

In the world of software development, size means different things to different groups of people. Those who specify functional requirements – and perhaps pay for the project, too – may conceptualize size in financial terms. Since project effort is a key component of cost, sizing in effort-units allows them to place the project in a cost-and-resource framework.

Software developers, while keenly aware of the effort required to complete their tasks, are more likely to describe size in terms of the things they have to produce to implement the requirements. Their sizing units are screens to be developed and modified, reports, database tables, Web pages, scripts, object classes, and a host of others.

To the software estimator, size quantifies what a project delivers or proposes to deliver. Concrete measures such as source lines of code or more abstract ones such as function points are the estimator's size units, and are indeed the ones needed to use a commercial estimating tool.

What is certain is that in software development, the word size may mean effort, programming artifacts, or elementary units of work depending on who is using the term. This is a potential source of confusion and miscommunication.

This article outlines a process for mapping requirements to intermediate units to elementary units of work, as shown in Figure 1, and uses the resulting output for estimating. The process is flexible and uses historical data to tune its algorithms.

Traditional Sizing and Estimating

Historically, software estimating has followed a pattern similar to the following:

- Requirements are broken down into software elements.
- Effort-hours for the tasks to create the software elements are estimated.
- The effort-hours are summed and a management reserve (fudge factor) is added to give an effort-estimate.
- Resources are leveled and a critical path is determined that allow project staff and duration to be estimated.

Unfortunately, this bottom-up approach is fraught with problems:

- It underestimates the overhead required and the non-software tasks associated with a larger project, often dramatically.
- Bottom-up estimating cannot be done effectively early in the project life cycle when bid/no bid or go/no go decisions are made and money, time, and staff are allocated to the effort. There is simply insufficient detail to determine all of the software elements, much less the project elements.
- It ignores the impact on schedule and effort of different sized teams. Schedule is simply effort divided by staff.
- It does not account for the non-linear impacts of time, cost, and quality constraints.
- It is not suitable for rapid, cost-effective, *what-if* analysis.

A critical element is missing from this approach and that element is project size.

An Alternative Approach to Sizing/Estimating

Parametric or model-based estimating takes the following different approach:

- It determines the size of the software elements breaking them down into common low-level software implementation units (IUs). (This will be discussed in the following section.)
- It creates a model-based *first cut estimate* using a productivity assumption (preferably historically based), the project size, and the critical constraints.
- It performs *what-if* modeling until an agreed-upon estimate has been created.
- It creates the detailed plans for the project.

Figure 2 illustrates this approach. Key to the success of this methodology is an accurate size and a productivity assumption that is consistent with the organization's capabilities.

Translating Requirements Into IUs

Customers have needs. These take the form of requirements that software must fulfill. Developers translate these requirements into intermediate units that they must create or modify to implement the requirements. These can be screens, programs, reports, tables, object classes, interfaces, etc. The list is fluid. Estimators must decompose the intermediate units into IUs to determine a size for estimating.

Figure 1: *Development Process*

Conceptually, an IU is the lowest level of programming construct that a software developer performs. It will vary in form depending on what is being developed. It could be setting a property on a Web form, indicating the data type of a field on a database table, or writing a line of procedural code. In each case it is the most elementary activity that the developer performs. Intermediate units are the tangible results of several or many IUs.

Two traditional size measures for estimating are source lines of code and function points. Both of these can work in some cases; however, each has limitations. The lines of code that a project generates are strongly influenced by the software languages used, individual coding style, and organizational standards. They are a measure of output that can be difficult to estimate.

Function points can be estimated from requirements and design documents but require training in the function point methodology and actual counting experience that many organizations lack. Function point counting is also a manual process that requires an investment of time and effort to perform. Although there are software tools that can capture the results of a function point count, there are none certified by the International Function Point Users Group as being capable of conducting the count.

An Alternative Sizing Approach

Here is a process for obtaining a size estimate that is conceptually simple, easy to implement, and encourages developer *buy-in*:

- Hold a facilitated session with the developers. Have them identify all of the intermediate units that they have to create. Determine what they physically have to do to create them. Ask if there are other things that they have to create on other projects. The purpose here is to establish a comprehensive list of the artifacts the developers may create. Good interviewing skills are the key to success here. Ask follow-up questions and keep asking if there is anything more. Developers may take some time to warm to this approach, but asking people to talk about themselves and what they do is a time-tested method of keeping a conversation going!
- For each item, have them define in quantifiable terms what makes that item simple, average, or complex. For instance, a simple screen might only have retrieval capability, while an average screen would also allow data entry. A complex screen would have update and delete capabilities, as well. Record the intermediate units in both effort-hours and IUs, which may be a ratio of effort at this stage. It is especially important to have several developers involved in this. Individual productivity can vary significantly between individuals, which influences their perspectives. Also, having the group of developers determine effort ranges will help balance overly optimistic and pessimistic estimates and help create buy-in.
- Construct a sizing worksheet that captures the results of the session. Figure 3 is a simple illustration of this concept.
- For a medium-size project with a small team, this process will normally take between four and six hours with between four and eight developers; this is where you get buy-in from the developers. Very large projects may well require additional time, but the method remains the same.

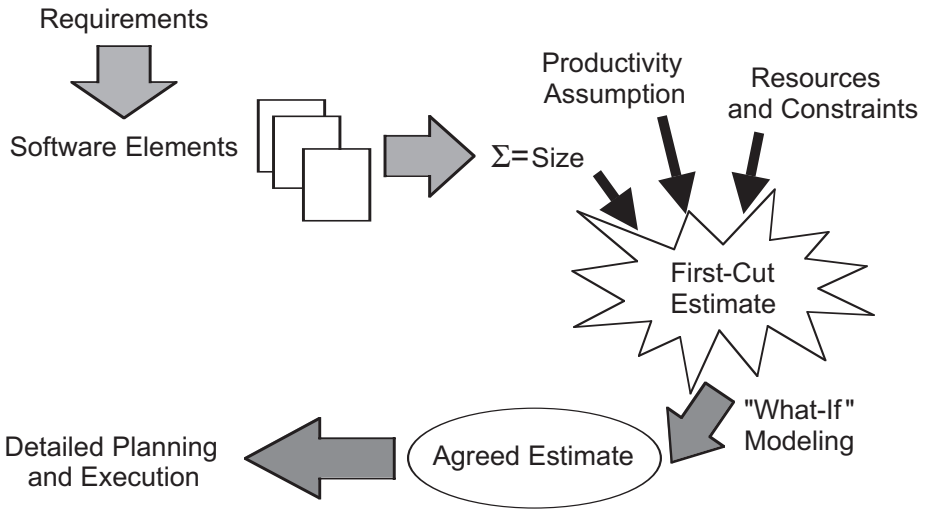

Figure 2: *Alternative Sizing and Estimating Approach*

Figure 3 is an example of a sizing spreadsheet. Using the intermediate units specified by the developers during the interview for this particular project type, data has been captured for a hypothetical project. The intermediate units that the developers have identified as being in their environment are in the first column. The second column contains the developers' estimate of the average hours required to create each intermediate unit. The IUs in

Figure 3: *Sizing Spreadsheet Example*

Intermediate Units	Effort Hours	IUs	Count	Total IUs	Total Effort
Forms - Simple	8	70		0	0
Forms - Average	15	170	8	1,360	120
Forms - Complex	30	400		0	0
New Report - Simple	13	140		0	0
New Report - Average	32	300	8	2,400	256
New Report - Complex	42	440		0	0
Changed Report - Simple	10	90		0	0
Changed Report - Average	24	250	4	1,000	96
Changed Report - Complex	31	320		0	0
Table Changes - Simple	5	60		0	0
Table Changes - Average	13	140	10	1,400	130
Table Changes - Complex	20	220		0	0
JCL Changes - Simple	1	12		0	0
JCL Changes - Average	4	50		0	0
JCL Changes - Complex	6	70		0	0
SQL Procedures - Simple	1	14		0	0
SQL Procedures - Average	10	140		0	0
SQL Procedures - Complex	20	225		0	0
Total Implementation Units				**6,160**	0
Total Effort Hours					**602**

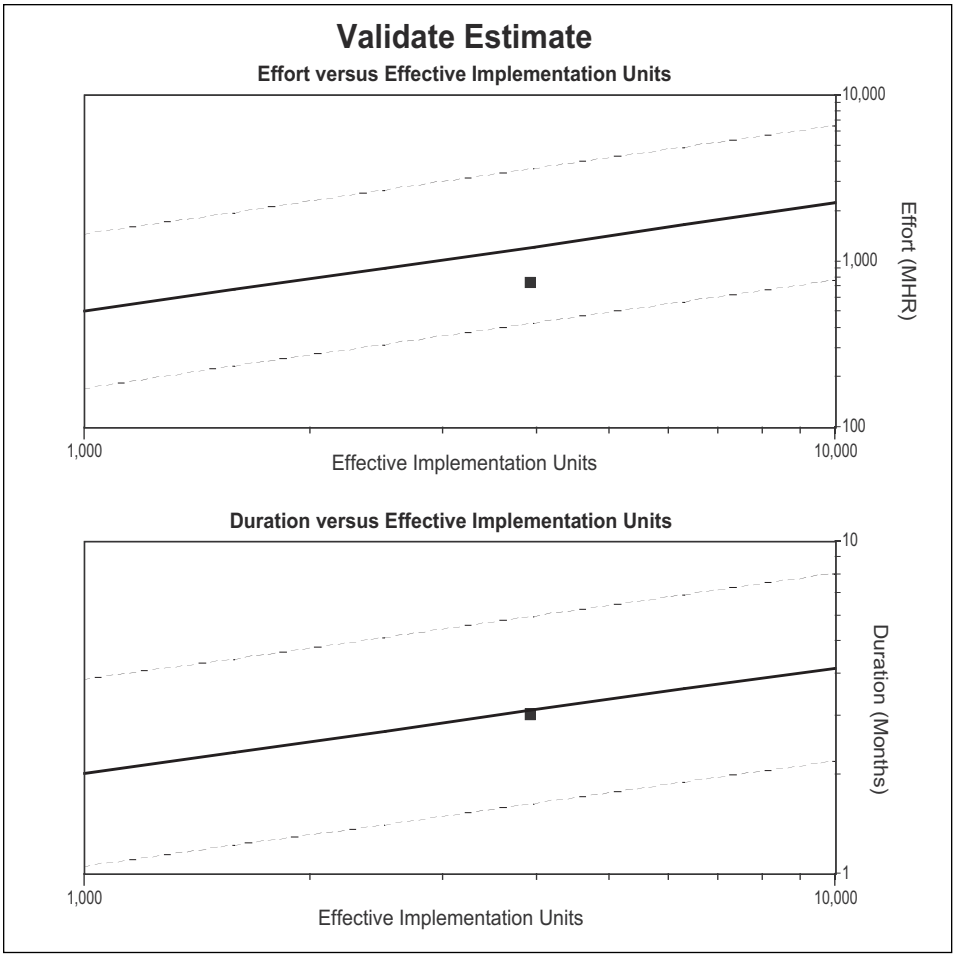

Figure 4: *Comparing an Estimate to Historical Data*

the third column are a weighting factor for each intermediate unit. If empirical data from other sizing spreadsheets is unavailable or if this is the first time this activity has been conducted, the IUs may simply be a multiple of effort. Column four contains the number of a particular intermediate unit that a project is estimated to have. The fifth column is the total estimated IUs for the intermediate unit. Column six is the total estimated effort-hours to create the intermediate units.

This is only a starting point and the IUs will be fine-tuned if required later in the process. If the developers have the project effort and intermediate units from a recently completed project, this is an excellent time to validate the estimated effort-hours on the worksheet. For instance in Figure 3, the total effort hours for the project were 602 to create the listed intermediate units. If the actual project hours of the project from which this was modeled were close to this, it lends credence to the effort estimates provided for the intermediate units. If not, it may indicate that adjustments need to be made to the effort estimates for the intermediate units or that there were intermediate units that should have been included in the project, or excluded.

Creating Estimating Templates

While interviewing the developers, it is important to have them define their project types. These may be as simple as small, medium, and large based on estimated effort hours. They may be platform-based such as Web, client server, or mainframe projects. They can also be application-specific or customer-specific.

Have the developers define what types of intermediate units are typical for each project type and a range of how many are normally found. Ask them to identify the effort range associated with each project type and identify typical durations. Tailor the estimating spreadsheets to each project type so they include only the intermediate units that project type is likely to have.

At this point the estimator can use the estimated size in IUs to create a template and calculate time, effort, and cost with a commercial parametric estimating tool.

Tuning the Process

The templates created in Figure 3 are starting points and will need to be fine-tuned. They are based on assumptions about the number of IUs per intermediate unit. How can this be refined? One method is to model completed projects using the sizing information captured on the templates as inputs to a parametric estimating tool. In this situation, project effort and duration are already known and there is an estimated size in IUs. The variable to be determined is the productivity parameter required to re-create an estimate scenario whose effort and duration match the completed project. This is a relatively easy thing to do with a parametric estimating tool. Even though solving a calculation for a missing variable will produce a result, it does not guarantee that the result is realistic. It is important to verify that the productivity parameter is reasonable when compared to industry data or organizational history.

Figure 4 demonstrates a method of comparing an estimate scenario to historical data to see if that scenario is internally consistent and reasonable. There are two graphs in Figure 4. Each has a set of trend lines calculated from a database of over 6,200 software projects. The darker line in the middle is the average. The dotted lines represent plus and minus one standard deviation, respectively. Note that a logarithmic scale is used to account for the non-linear relationship between project size and effort or duration. The X axis of both charts is project size in IUs. The Y axis on the top chart is project effort in manhours.

In this case, the effort-hours for the project (represented by the square) are slightly below the average line for similarly sized projects. The Y axis of the lower chart is project duration in calendar months. This project falls right on the average line. For this estimate scenario, both effort and duration are historically consistent with similar sized projects.

The productivity parameter used is also historically consistent[1]. If the effort were very high compared to the trend lines, it could indicate that the IUs were understated (too much effort for the amount of output). Extremely low effort compared to the trend lines would suggest that the IUs were overstated.

Similar comparisons apply for the bottom graph, too. If after modeling several projects, effort, duration, or both are consistently very high or very low, then it is a strong indication that the number of IUs for some of the intermediate units requires adjustments.

Sizing templates can be further refined as projects complete. There is one final word of caution to consider when modeling projects: The projects

should be as normal and representative of the work usually done as possible. The intent is to build a model that reflects how work is usually done. Projects with cherry-picked teams or ones that suffered from extreme schedule pressure or rework due to requirements changes are not good candidates to model. They will only skew the results.

Benefits

As Frederick Brooks [1] warned us nearly 30 years ago, there is no silver bullet. This approach to sizing may not be the best fit for every software development situation. But, it will work in many situations and has some real benefits:

- It speaks the developer's language. It describes the system in the components that developers work with: screens, reports, tables, programs, and Web pages. This improves communication.
- It involves the developers in the estimating process creating buy-in and reducing the chance of obtaining bogus data.
- It is adaptable. It allows new tools and components to be incorporated easily.
- It is an excellent way to get a handle on a new technology. It provides the ability to articulate what and how developers build a product.
- It is applicable to many different development paradigms, some of which have been difficult to estimate with parametric estimating tools:
  - o Enterprise Resource Planning (PeopleSoft and SAP [Systems, Applications, Products]).
  - o Rational Unified Process.
  - o Traditional Development.
- It can (and should) be tuned on actual project data.

If you are running into roadblocks when estimating the size of your application development projects, give this method a try. You might be pleasantly surprised by the cooperation that you receive from the technical staff, and the increased value that is attached to your end-product estimates.◆

Reference

1. Brooks, Frederick. The Mythical Man-Month: Essays on Software Engineering. Addison-Wesley, 1 Jan. 1975.

Note

1. Estimating tools look at productivity from different perspectives. What is important is that however productivity is measured, there needs to be a method in place to validate it for reasonableness against organizational or industry data.

About the Authors

Lawrence H. Putnam is the founder and chief executive officer of Quantitative Software Management, Inc., and a developer of commercial software estimating, benchmarking, and control tools known under the trademark SLIM. He served 26 years on active duty in the U.S. Army and retired as a colonel. He has been deeply involved in the quantitative aspects of software management for the past 30 years. He is the co-author of five books on software estimating, control, and benchmarking. He is a member of Sigma Xi, the Association for Computing Machinery, the Institute of Electrical and Electronic Engineers (IEEE) and the IEEE Computer Society. He was presented the Freiman Award for outstanding work in parametric modeling by the International Society of Parametric Analysts. Putnam has a Bachelor of Science from the United States Military Academy and a master's degree in physics from the Naval Postgraduate School.

Quantitative Software Management, Inc.
2000 Corporate Ridge STE 900
McLean, VA 22101
Phone: (703) 790-0055
Fax: (703) 749-3795
E-mail: larry_putnam_sr@qsm.com

Douglas T. Putnam is the managing partner of Professional Services at Quantitative Software Management, Inc. (QSM). Putnam has over 24 years of experience in the software measurement industry. He has written and lectured extensively throughout the world and has participated in more than 200 estimation and measurement engagements in his career at QSM. QSM is the supplier of the trademarked SLIM suite: SLIM-Estimate, SLIM-Master Plan, SLIM-Control, SLIM-Metrics.

Quantitative Software Management, Inc.
2000 Corporate Ridge STE 900
McLean, VA 22101
Phone: (703) 790-0055
Fax: (703) 749-3795
E-mail: doug_putnam@qsm.com

Donald M. Beckett is a consultant for Quantitative Software Management with more than 20 years of software development experience, including 10 years specifically dedicated to software metrics and estimating. Beckett is a Certified Function Point Specialist with the International Function Point Users Group and has trained over 300 persons in function point analysis in Europe, North America, and Latin America. He was a contributing author to "IT Measurement: Practical Advice from the Experts." Beckett is a graduate of Tulane University.

Quantitative Software Management, Inc.
2000 Corporate Ridge STE 900
McLean, VA 22101
Phone: (703) 790-0055
Fax: (703) 749-3795
E-mail: don_beckett@qsm.com