Estimating With Objects - Part VII

Contents

This column is the seventh in a series about estimating. The first was in the July 1996 issue. The most recent several columns have focused on estimates based on lines of code (LOC). There are, however, several types of source code as well as many other important product types. Examples are new, modified, and reused code, as well as documentation, screens, reports, and files. The principal question addressed by this column is: What are the most important product types to consider, and how do you estimate them?

The prior columns in this series gave an overview of estimating and defined some of the steps in making size and resource estimates. If you have not read these earlier columns, you should look at them first to understand the context for this discussion and to see how these various estimating topics relate. To repeat what I have said in previous columns, the estimating method described here is called PROBE. If you want to quickly learn more about PROBE, you should read my book *A Discipline for Software Engineering*, from Addison Wesley. This book introduces the Personal Software Process (PSP)^{SM}, which is an orderly and defined way for software engineers to do their work.

This column continues the discussion of how to make software estimates. To make a project plan, you need a resource estimate and, to estimate resources, you need to estimate the size of the product you plan to build. Also, to make a good size estimate, you need historical data on the sizes of the programs you have previously written. This and the previous columns describe how to gather these data and how to use them to make the size and resource estimates.

The principal kinds of code

When writing programs, engineers generally write some new code, modify some existing code, delete some code, and reuse code from prior programs. While each of these activities involves development effort, the amount of effort can vary considerably. For example, you might take about 5 hours to develop 100 new LOC, while it might take only about 10 to 20 minutes to reuse

100 LOC from a previous program. Similarly, depending on the complexity of the program and the sophistication of the changes, modifying 100 LOC in an existing program could take 15 to 30 hours or more, but deleting 100 LOC from the program would likely only take a few minutes.

While all these activities contribute to the finished product, they all take varying amounts of time. The question then is: Which of these activities do you need to consider and why?

New and changed LOC

Since few people have data on their relative productivities when producing different kinds of LOC, the general practice has been to only count new and changed LOC. For example, if you planned to develop a program estimated to have 350 new LOC, 75 modified LOC, 25 deleted LOC, and 500 reused LOC, you would just count the new and modified LOC. Thus, you would count 350 + 75 = 425 new and changed LOC. While this only provides a crude measure of the amount of work to be done, it is generally adequate, at least as long as the mix of new, modified, and reused code does not vary too widely.

This does not mean that the reused and deleted code are not important. It also does not imply that reuse should not be encouraged. The reason to only count the new and changed LOC is because that is the activity that takes the most time. Clearly, to make an accurate resource estimate, you must concentrate on those activities that take the most time. Also, since it takes relatively little time to delete or reuse code, the estimating errors introduced by ignoring such efforts are generally small. In fact, when you use the linear regression method (discussed last month), you automatically include a factor for the average percent of development effort spent deleting and reusing code. If, for some reason, you need to precisely account for the mix of new, modified, deleted, and reused code, a later section of this column describes the multiple regression method which does this.

Motivating reuse

Another objection to not counting reused code in an estimate is that it does not motivate programmers to reuse programs. While I do not believe this is a serious problem, the principal objective in making an estimate should be to achieve accuracy. Even considering the fact that estimating methods have motivational consequences, the principal estimating concern should be accuracy. Where motivation is a problem, motivational actions are called for. Examples would be measuring the amount of reuse, establishing reuse goals, and giving awards or prizes. It makes no sense, however, to screw up the estimating process to achieve a motivational objective.

Finally, let me assure you that I am not against reuse. In fact, after the PSP, I am convinced that reuse is the most-effective way to increase productivity, reduce product development cycle time, and improve quality. If you have a high quality development process, you should work to maximize reuse. There is, however, one caution. If your organization produces poor quality software, reuse is not a good idea. The resulting proliferation of defects will cost more than reuse could possibly save.

Other product types

We have now discussed four types of code: new, modified, reused, and deleted. Are there other product types that should be considered? Here, again, the issue concerns estimating accuracy. That is, by considering other product types, can we improve estimating accuracy? The topics to consider in answering this question are the principal product types involved in this project, the development process, and the customer requirements. For example, when developing programs for the U.S. Department of Defense, documentation is generally a major expense. Often, in fact, there are several pages of documents for every program LOC. Documentation costs could, in fact, even exceed all other software development and testing costs.

The way to decide what product types to estimate is to consider all the work products and determine which take the most development time. Then, the activities that consume little time can generally be ignored, or at least estimated by rule of thumb. The big swingers, however, need to be estimated directly.

A database example

For example, suppose you were estimating the development of a new database. This work would likely include a requirements review, a study of the existing databases, examining the sources for the data, and determining the format constraints. The principal development work would involve designing the file and record formats and developing data entry, verification, and reporting methods. Developing source LOC would probably not be a major part of the job. In fact, the total development time would likely be more closely correlated to the number of file and record types than to the LOC produced.

Fourth generation languages and program generators

Another important issue concerns program generators. For example, some 4th generation languages generate a great deal of source code for a relatively few key strokes. You do not want to count the generated LOC because it is produced by the program generator. To use any productivity-based estimating method, like regression, you must concentrate on the products produced by the engineers. Thus, you might count screens, reports, and manually entered source statements. Always, however, test the measures to ensure that they correlate with development time. If they do not correlate, then using an estimate of the amount of that product would be misleading and could even reduce estimating accuracy. The key is to only use product measures that correlate with development effort. If you use other measures, they will likely make your estimates worse.

Calculating correlation values

Since correlation is so important, I digress here to briefly describe the correlation calculation.

$$r(x,y) = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{\left[n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2\right]\left[n\sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2\right]}}$$

Here, when you have two sets of data, x, and y, with n values for each, and you want to determine their correlation, use this formula to calculate r, the correlation. For example, x might be the new and changed LOC and y might be the development hours for each of n projects.

For estimating purposes, r^2should be greater than 0.5. While statisticians talk about the significance of a correlation, as long as you have seven or more data points, any correlation with an r^2greater than 0.5 will be significant. If you only have five data points, then you need an $r^2 \geq 0.6$If you have fewer than five data points, don't even worry about correlation; get more data! When the correlation is significant, the x and y values are closely enough related for estimating purposes.

If you want a more detailed explanation of these calculations, see almost any statistics text or Appendix A of my book A Discipline for Software Engineering.

Estimating strategy

Since you will often have to estimate many different kinds of work, and since you will rarely have good product size estimates for many of these activities, it is generally a good idea to develop estimating factors. For example, you might determine that documentation time averages 45% of design hours, or that the quality assurance effort averages 6% of total code and test time. Whatever the values, it is important to determine them for your organization. Generally, such ratios for other organizations will not be very accurate for your organization and using them could be misleading. When in doubt, get some data and see!

In summary, the strategy is as follows:

- Directly estimate the most important parts of the job.
- Identify the other important development activities.
- Obtain data on the percentage of total effort spent on each of these activities.
- Finally, using these historical factors, estimate the amount of each activity required for this job.

While this is not a foolproof method, and while it is always preferable to have the actual working groups estimate their own work, this is not always possible. Such historical estimating factors can also be a big help in reviewing estimates for reasonableness. When a new estimate has a substantially different pattern from the historical data, it is a good idea to look more closely. Often, the estimators will have forgotten something, or they may have made some poor assumptions.

Estimating for multiple job types

Suppose you have been having trouble with inaccurate estimates and conclude that the reason is the wide variation in the mix of product types. You would like an estimating formula to account for these multiple code types. Something like:

Development Hours = $\beta_0 + \beta_1$(New LOC) + β_2(Modified LOC) + β_3(Reused LOC)

Similarly, your work could involve a 4th generation language and LOC might not be a major consideration. Here, you might seek a formula like:

Development Hours = $\beta_0 + \beta_1$(No. of Files) + β_2(No. of Screens) + β_3(No. of Reports)

With either of these formulae, if you knew the values of the βs, and you had an estimate of the amount of each product type, you could estimate total development time. These formulae are called multiple regression.

While the calculations to find the βvalues are not difficult, the explanation of how to do it is a bit too lengthy for this column. If you wish to find out more about using multiple regression in estimating, consult my book, A Discipline for Software Engineering, from Addison Wesley. Multiple regression is discussed in Chapter 8 and in Appendix A.

Cautions on multiple regression

One of the principal problems with multiple regression is that you need a great deal of data. You also need to be careful in using multiple regression because it is easy to get unreasonable values for the βs. This is highly likely if you do not have enough data or if the size data do not correlate well with development effort.

It is rarely a good idea to use sophisticated estimating methods until you have mastered simpler techniques. I suggest you start by learning and using linear regression. After you have gained considerable experience and have a substantial amount of data, multiple regression could be a useful next step.

Next month's topics

We have now covered the basic issues of estimating with various product types. The next columns deal with estimating accuracy, scheduling, and tracking.

An invitation to readers

In these columns, I discuss software process issues and the impact of processes on engineers. I am, however, most interested in addressing issues you feel are important. So please drop me a note with your comments and suggestions. Depending on the mail volume, I may not be able to answer you directly, but I will read your notes and consider them when I plan future columns.

Thanks for your attention and please stay tuned in.

Watts S. Humphrey
watts@sei.cmu.edu