Estimating With Objects - Part IV

Contents

This column is the fourth in a series about estimating. This month, we discuss gathering, categorizing, and using object size data. The first column in this series was in July, and it gave an overview of estimating. The August column talked about software size, and last month's column introduced the subject of proxies. If you have not read these columns, you should look at them first to understand the context for this discussion and to see how the various estimating topics in these columns relate. To repeat what I said in the previous columns, the estimating method described here is called PROBE. If you want to quickly learn more about PROBE, you should read my book *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 size 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. Finally, to make a good size estimate, you need historical data on the sizes of the programs you have previously written. Last month, we introduced the subject of proxies. A proxy is a substitute, or alternate that you can use to help make size estimates. To use objects as proxies, however, you need historical data on object sizes. This column describes these data, how to gather them, and how to use them.

The data needed for size estimating

Suppose you were estimating a new program and had concluded that it would have 11 objects. How would this information help you, and what data would you need? You would first examine each of these planned objects to understand their general characteristics. Then, from your historical data, you would relate these new objects to the object data and judge how they compare. Finally, you would estimate the likely size of the new objects based on the known sizes of the objects these new objects most closely resemble.

The PROBE method helps you do this by showing how to divide data on the objects you have developed into type categories. It also shows how to determine the size distributions of these object types and how to present these data so you can conveniently use them in estimating.

Gathering object data

First, object size is a matter of personal style. The sizes and number of methods in a C++ class that you develop will likely differ from what I develop. Neither of us is right or wrong, we just have different programming styles. Thus, even for the identical program functions, the sizes of the programs we develop will differ.

For example, from the identical specifications, 32 engineers wrote an average of 242 LOC to develop a program to count C++ LOC. The median LOC for these programs was 176, the maximum LOC was 582, and the minimum was 87. The development times also varied from a little over two hours to over 18 hours. Most of these engineers had several years of industrial experience and they all said they knew how to write programs in C++.

Since different engineers develop programs differently, you need to use data on the programs you develop. If you don't, you are not likely to get accurate estimates. Also, since the sizes and development times for the objects you develop will likely change as you gain experience, you should periodically examine your development data. This will ensure that you are using data that relate to the way you currently develop software.

Gathering object LOC data

To gather object LOC data, you need to count the total LOC in each object, the number of methods in the object, and the LOC in each method. If you are an active programmer, you will probably have a fairly large collection of classes, objects, and methods that you can count. Actually counting their LOC may, however, not be easy. While you could do this by hand, that is tedious and inaccurate. You will thus want a program to count LOC. Unfortunately, commercial LOC counting programs are not available to gather the kind of data you need.

Writing object LOC counters is not an impossible chore. In fact, the program the 32 C++ programmers wrote that I mentioned above was a C++ object LOC counter. While developing such a program is a non-trivial job, it should only take a few hours. Both the average and median development times for these 32 engineers was under 5 hours. If you want to know how to write such a program, see Chapters 4 and 5 and read about programs 2A and 3A in Appendix D of my book Discipline for Software Engineering, from Addison Wesley.

Analyzing and presenting object data

Once you have counted the LOC in each object and method, you need some way to relate these data to the sizes of new program objects. Ideally, you would like to characterize the new objects in terms of the ones for which you have data. There are several parts to doing this. First, break the historical object data into object categories, or types. The types you pick should be meaningful for the work you do. In my C++ work, I have data on 98 methods from 20 objects in

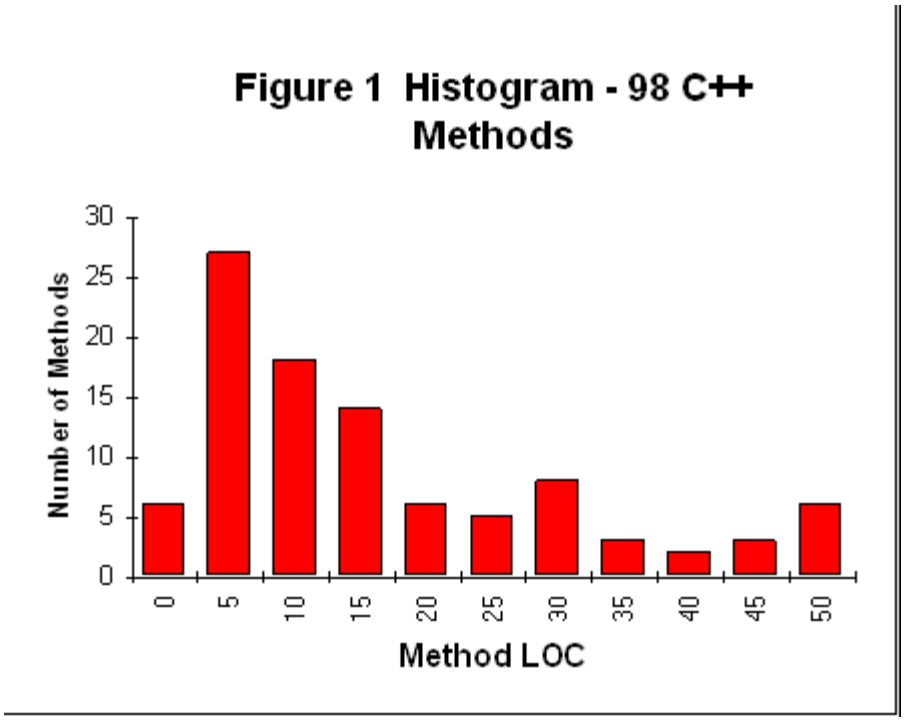
5 classes. Engineers who spend more of their time writing programs could presumably have a great deal more data than that.

To illustrate how to establish object type categories, I divided my C++ data into six types: calculation, data, I/O, logic, set-up, and text. Depending on the types of programs you write, you will presumably have similar categories. You must, however, have a reasonable number of methods in each category. Then you will likely have enough data in each category to provide useful estimating guidance. If you don't have enough data, you should keep the number of object types small enough so you have a minimum of 6 or more methods in each category.

The categorizing step determines the size ranges for each object or method type. With these size ranges, you would know, for example, that a large calculation object historically had 24.66 LOC per method. Thus, if you had a new calculation object that you felt would have 5 large methods, the object would have an estimated size of 123 LOC.

The problem, now, however, is to determine from your historical data the sizes of large, medium, or small methods of each type. Actually, in PROBE, we use five size categories: very small, small, medium, large, and very large. These provide a convenient framework for judging the sizes of new objects.

The size distribution problem



If object size data were normally distributed, like people's heights for example, there would be no problem in deciding what was a medium, large, or very small object. Unfortunately, object

and method LOC are not normally distributed. They tend to bunch up, particularly at small method sizes. The reason is that we have no negative object sizes. Such bunching is not a problem with peoples heights because average heights are much larger than zero. With objects and methods, however, many sizes will likely be quite small. At least they were for my programs.

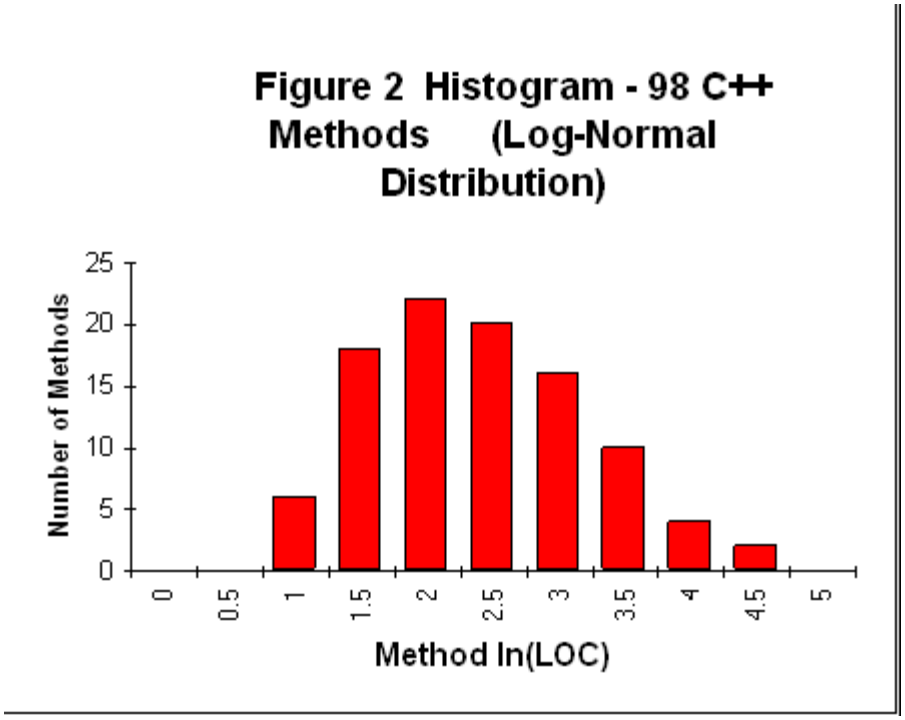


Figure 1 shows a histogram of the LOC of my 98 C++ methods of all types. As you can see, the distribution is far from normal. One way to handle this problem is to use what we call a log-normal distribution. Here, we first take the natural logarithms of the size data to get a more normal distribution. Figure 2 shows how taking the natural logarithm of the sizes changes the histogram for my C++ methods. The distribution now more closely resembles the traditional bell-shaped normal curve.

Next, with the log-normal data, we calculate the average and standard deviations of these method size data. Then, still using the log-normal data, we find the log-normal values that are one and two standard deviations above and below the log-normal average. This gives us the log-normal size ranges we seek. For example, the very small size would be two standard deviations below the average, and the small size would be one standard deviation below the average. Finally, we convert these log-normal size range values to LOC terms by taking their natural exponent, or inverse logarithm. This gives the size ranges we will use in estimating.

The data for my 98 C++ methods are shown in Table 1. Note that when we did the distribution calculations on the LOC data, we get a negative value of -17.8 LOC for the VS size category. This of course makes no sense. The Log normal distribution on the right of the table is far more useful. The following example shows how these values were obtained:

Very Small[ln(LOC)] = Average[ln(LOC)] - 2*Std. Dev.[(ln(LOC)]

= 2.651- 2*0.805

= 1.041

Very Small(LOC) = exp(1.040) = 2.831 LOC

Table 1. Size Ranges for 98 C++ Methods

	Value	LOC	ln(LOC)	Final	
	Average	19.857	2.651	14.166	
	Std. Dev.	18.832	0.805	2.237	
	Very Small	-17.808	1.041	2.831	
	Small	1.025	1.846	6.333	
	Medium	19.857	2.651	14.166	
	Large	38.690	3.456	31.691	
	Very Large	57.522	4.261	70.896	

The values for my C++ object types are shown in Table 2.

Table 2. Size Ranges for C++ Object Types - (LOC/Method)

Category	Very Small	Small	Medium	Large	Very Large
Calculation	2.34	5.13	11.25	24.66	54.04
Data	2.60	4.79	8.84	16.31	30.09
I/O	9.01	12.06	16.15	21.62	28.93
Logic	7.55	10.98	15.98	23.25	33.83
Set-up	3.88	5.04	6.56	8.53	11.09
Text	3.75	8.00	17.07	36.41	77.66

Using object size data

With these object size data, you start to make a size estimate by producing a conceptual design for the new product. We will discuss more about how to do this next month. Then you identify all the objects in this conceptual design, and categorize these objects into the types for which you have historical data. Next, judge how many methods each object will have. In fact it is even helpful to name each method, if you can. Now, judge where each object and method falls in the size scale from very small, small, medium, large, or very large for objects and methods of the same type. Finally, from your historical data, calculate the estimated LOC for each method and object.

While you could merely estimate the total number of LOC in the objects, I have found it easier to estimate the number of methods and the LOC per method. This may seem like a lot of detail, but going into detail is the only way to consistently make good estimates. We will talk more about the detail problem next month. The last step is to use these object data to estimate the total program size and the development time. To find out how to do this, you will have to read next several month's columns.

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