5

Software Estimating

This chapter addresses size estimating. It starts with a brief description of conceptual design and then covers proxy-based estimating. Chapter 6 describes the PROBE method for using proxies to make size and time estimates. As shown in Figure 4.1 in Chapter 4 (see p. 63), the PSP planning process starts with the conceptual design. This provides the basis for the size and resource estimates. At the outset, you will know relatively little about the planned product, so your estimates will be least accurate. Because you must presumably commit to a delivery date, however, you must make the best estimate that you can. This chapter describes how to identify the parts of a planned product and estimate their sizes.

5.1 Size Estimating Principles

In principle, estimates are made by comparing the planned job with previous jobs. By breaking planned products into multiple smaller parts and comparing each part with data on similar parts of prior products, you can judge the size of the new product. This strategy works well for estimating almost any kind of development work. However, it requires data on the products you have developed and the work required to develop them. You also need a method for using historical data to make the estimates.

This divide-and-conquer estimating strategy has the advantage of scaling up. That is, once you have learned to make accurate estimates for the PSP exercises, you can use the identical methods to estimate larger jobs. The only difference is that you will have to divide bigger jobs into more parts or get data on larger parts.

In planning a PSP project, I use the term *product* to refer to the end item produced by the project and I use the term *parts* to refer to the elements of a product, whatever you plan to call them. Depending on the type of development, the parts could be systems, subsystems, components, classes, procedures, databases, or whatever term best applies to that particular product. For example, when I estimate a C++ program, the parts are classes. With Object Pascal, they would be objects, and with database work, they would be fields, tables, or queries. If the product's parts have even smaller elements, such as methods, functions, or procedures, I call them *items.*

5.2 The Conceptual Design

When first making a plan to develop a product, you may generally understand the requirements but little else. The estimating challenge is then to predict the size of this vaguely defined product and the time required to develop it. Because no one can know in advance how big a planned product will be or how long it will take to develop, estimating will always be an uncertain process. Therefore, you need to use the best product requirements and historical data you can get.

With the PSP, you make a size estimate first and then make the development time estimate. For the size estimate to be reasonably accurate, it must be based on an initial design concept, and that conceptual design must reflect the way you plan to build the product. As noted in Chapter 4 (see p. 63), the conceptual design defines a preliminary design approach and names the expected product parts and their functions. Do not produce the complete design during planning; just postulate the parts you will need and the functions they will perform.

For an accurate estimate, you must refine the conceptual design to the level of the parts you know how to build. Next, check for historical data on similar parts. If a part does not resemble anything in your historical records, see if you have refined the conceptual design to the proper level. Assuming you have a reasonably complete and well-structured parts database, you will likely find that the part is a composite of several more basic parts. If so, refine it into those more basic parts. If a conceptual part is at the right level and still does not look like any of your existing part data, estimate it as the first of a new kind of part. In doing so, compare this part to other product parts in the database to get a feel for their relationships and then make your best intuitive estimate of its size. Then, after you have completed the development work, measure the size of the new parts and enter these data into your database for use in estimating future work.

5.3 Proxy-Based Estimating

Consider the example of construction. In home building, the number of square feet of living space provides a basis for estimating construction costs. Few people, however, can visualize the house they want in terms of square feet. They think of number of bedrooms and bathrooms. Software estimating has similar problems. If we could judge the number of database relations or LOC for a product requirement, we could probably make a pretty good estimate of that product's size. Unfortunately, few people can directly judge how many LOC or database elements it would take to meet a product requirement.

What is needed is some proxy that relates product size to the functions we can visualize. A proxy is a substitute or a stand-in. Assuming it is easier to visualize the proxy than the size measure, the proxy can help you to judge a product's likely size. Examples of proxies are classes, tables, fields, or screens. The generalized proxy-based estimating process is illustrated in Figure 5.1.

Selecting a Proxy

The criteria for a good proxy are as follows:

- The proxy size measure should closely relate to the effort required to develop the product.
- The proxy content of a product should be automatically countable.
- The proxy should be easy to visualize at the beginning of a project.
- The proxy should be customizable to the needs of each project and developer.
- The proxy should be sensitive to implementation variations that affect development cost or effort.

These points are discussed in the following sections.

Related to Development Effort

To be useful, a proxy must have a demonstrably close relationship to the resources required to develop the product. By estimating the relative size of the proxy, you can accurately judge the size of the planned product. To determine the effectiveness of a potential proxy, obtain data on the products that you have developed and

31422 05 pp. 069-084 r3ah.ps 2/9/05 5:21 PM Page 72

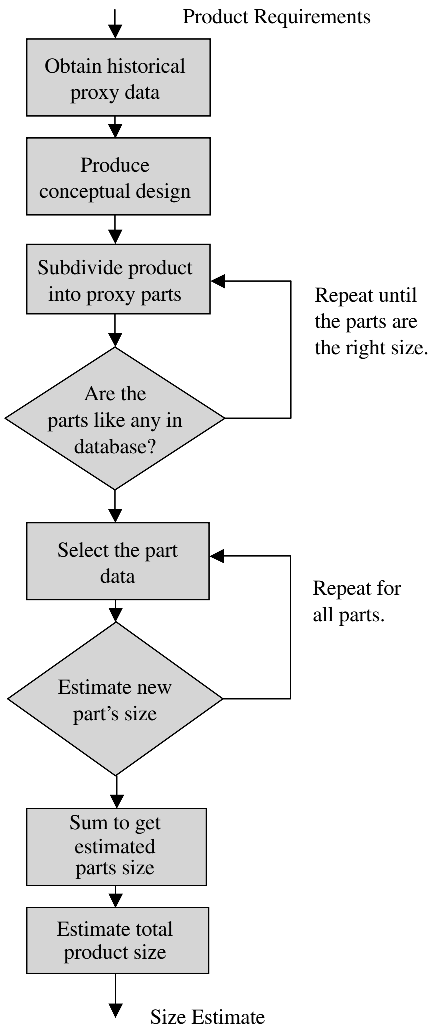
Product Requirements
Obtain historical proxy data
Produce conceptual design
Subdivide product into proxy parts
Repeat until the parts are the right size.
Are the parts like any in database?
Select the part data
Repeat for all parts.
Estimate new part's size
Sum to get estimated parts size
Estimate total product size
Size Estimate

FIGURE 5.1 PROXY-BASED SIZE ESTIMATING

compare the proxy values with their development times. Using the correlation method, determine if this or some other proxy is a better predictor of product size or development effort. If the proxy does not pass this test ($|r| >= 0.7$), find another one that does. You can quickly determine this by examining the data in a scatter plot, like that shown in Figure 3.1 (see p. 39) and in Figure 5.2. The correlation

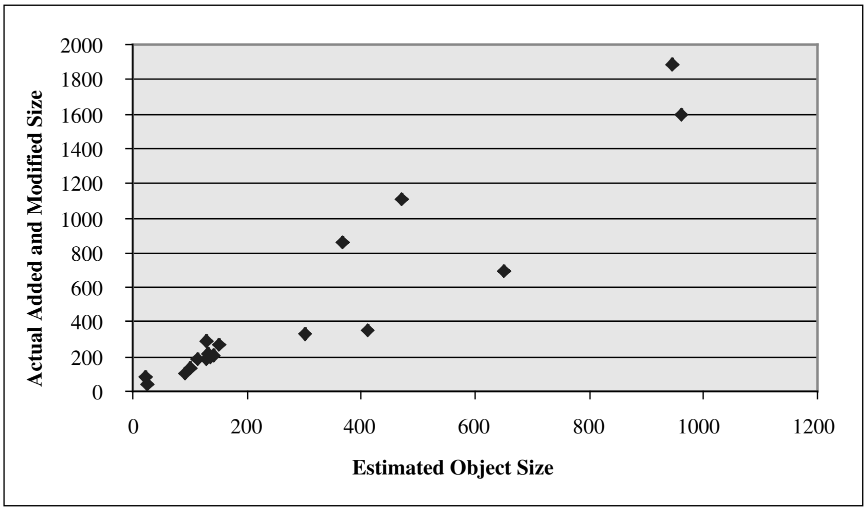

FIGURE 5.2 PART SIZE VERSUS ACTUAL SIZE (18 PASCAL PROGRAMS)

coefficient *r* is calculated by most PSP and TSP tools, but if you want to calculate it yourself, the formula for the correlation calculation is shown in Box 3.1 in Chapter 3 (see p. 36).

Because it is possible to get very high correlations with small amounts of data, statisticians use a significance measure to indicate the likelihood that a relationship either has predictive value or occurred by chance. The significance calculation is shown in Chapter 3 in Box 3.2 (see p. 37).

The Proxy Content Is Automatically Countable

Because estimating accuracy improves with more and better data, you will want a lot of proxy data. This suggests that the proxy should be a physical entity that can be precisely defined and automatically counted. If you cannot automatically count the proxy content of a product, there is no convenient way to get the data you will need to improve your estimating accuracy.

Easily Visualized at the Beginning of the Project

If the proxy is harder to visualize than the number of hours required to develop a product, you may as well estimate the hours directly. The usefulness of a proxy thus depends on the degree to which it helps you to visualize the size of the planned product. This in turn depends on your background and preferences. There will likely be no best proxy for all people or purposes. With suitable historical data, you could even use several different proxies in one estimate. The multiple-regression method, usually introduced with the final PSP exercise program, can be helpful for this purpose.

Customizable to Your Project's Needs

Much of the difficulty people have with estimating methods results from using data from one group to plan work by another person or group. It is important, therefore, to use data that are relevant to your particular project. This suggests that you build a size and resource database for each type of product you develop. If the proxies you select are not suitable for some kinds of work, examine your data to identify more suitable proxies. After writing several programs, you should have enough data to identify useful proxies.

Sensitive to Implementation Variations

The proxies that are most easily visualized at the beginning of a project are application parts such as classes, inputs, outputs, database fields, forms, and reports. However, a good development estimate requires parts that closely relate to the product to be built and the labor to build it. This requires that proxy data be available for each implementation language, design style, and application category.

Possible Proxies

Many potential proxies could meet the criteria outlined above. The function-point method is an obvious candidate because it is widely used. Many people have found function points to be helpful in estimating resources. However, because function points are not directly countable in the finished product, there is no simple way for developers to get the data needed to improve their function-point estimating accuracy.

Other possible proxies are classes, database elements, forms, scripts, and document chapters. The data gathered during the PSP research show that database elements, classes, and document chapters generally meet the proxy criteria. For database work, developers have found that a count of all database elements is a reasonably accurate predictor of development effort. Example elements would be fields, tables, queries, and records.

Classes as Proxies

The principles of object-oriented design suggest that, for many kinds of programming, classes would be good estimating proxies. An application's conceptual parts would then be items that exist in the application environment. For example, in an automobile registration system, these might include automobiles, owners, registrations, titles, or insurance policies. In the conceptual design, you would select program classes that model these real-world conceptual parts. These highest-level product classes could then be visualized during requirements analysis. Classes thus potentially meet one of the requirements for a proxy.

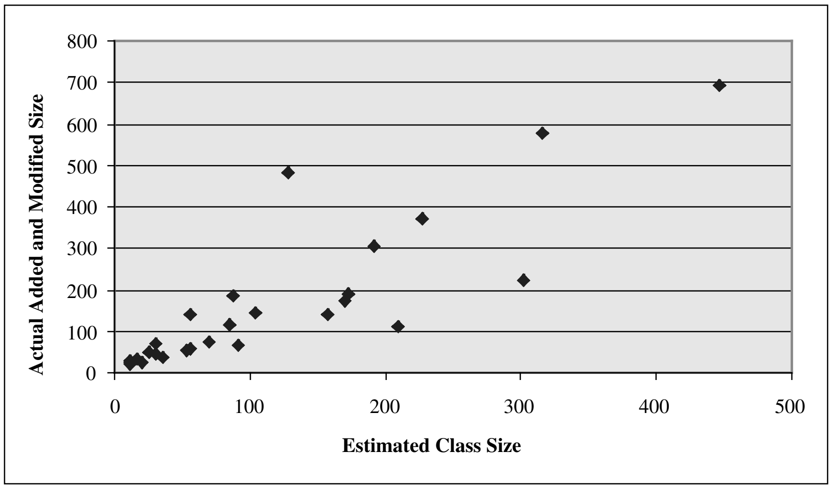

FIGURE 5.3 ESTIMATED VERSUS ACTUAL SIZE (27 C++ PROGRAMS: $r = 0.83$)

To determine whether class is a good size proxy, examine your historical data. Figure 5.2 shows the relationship between estimated object size and actual added and modified size for a family of 18 Object Pascal programs that I wrote during the PSP research work. Figure 5.3 shows similar data for a family of 27 C++ programs. Figures 5.4 and 5.5 compare these estimated part sizes (in LOC) with the hours spent developing these same Pascal and C++ programs. In both cases, the correlation and significance are very high. Because total program LOC correlates to development hours, classes thus meet the requirement that they closely relate to development effort.

Finally, as discussed in Chapter 3, classes are physical entities that can be automatically counted. Assuming that you use an object-oriented design methodology and programming language and that you gather size data on the classes you develop, classes will meet all of the criteria for an estimating proxy.

5.4 Using Proxies in Estimating

To use proxies in estimating, divide your historical size data into categories and size ranges. The construction example illustrates why this is important. When estimating the square feet in a house, a builder considers big rooms and small rooms and how many of each the buyer wants. The builder does this by room categories;

31422 05 pp. 069-084 r3ah.ps 2/9/05 5:21 PM Page 76

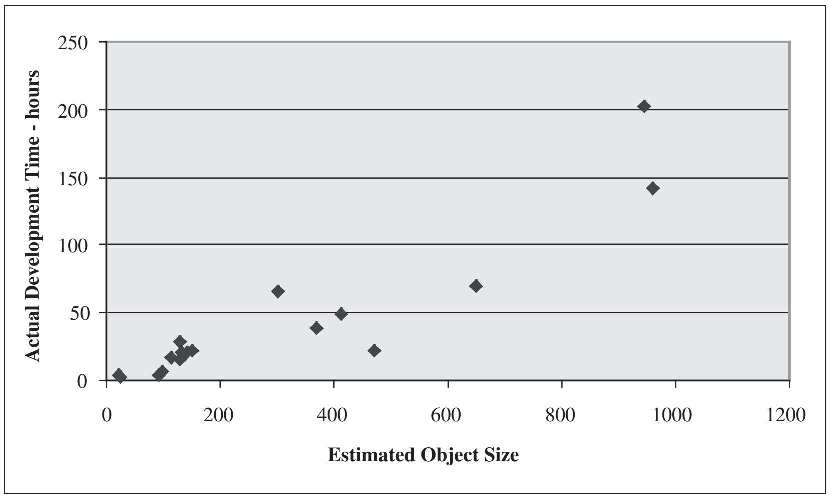

FIGURE 5.4 PASCAL ESTIMATED SIZE VERSUS ACTUAL HOURS ($r = 0.915$)

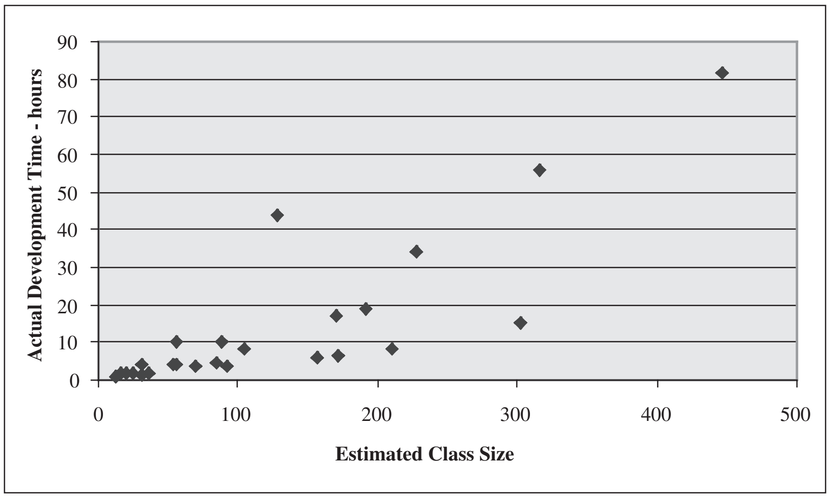

FIGURE 5.5 C++ ESTIMATED LOC VERSUS ACTUAL HOURS ($r = 0.829$)

for example, a big bathroom would likely be smaller than even a small family room.

Similarly, in estimating how many LOC a part contains, group the data into functional categories. Then estimate how many parts of each category are needed for the product, and the relative size of each part. Finally, decide how each part relates to its category, e.g., very small, small, medium, large, or very large. The Pascal and C++ categories I used with the PSP are shown in Table 5.1. At the top of Table 5.1, the C++ classes are listed in six categories. These data show that a medium-size C++ class item for mathematical calculations would have 11.25 LOC, while a very large calculation part would have 54.04 LOC.

The size of the parts you develop is also largely determined by your personal programming style. Some people prefer to group many items (or functions) into a few parts, while others prefer to create many parts with relatively few items. The estimating data should accurately reflect the languages and practices you use in developing software. Therefore, it is a good idea to use part size data derived from your own work.

TABLE 5.1 PARTS LOC PER ITEM

C++ Class Size in LOC per Item					
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

Object Pascal Object Size in LOC per Item					
Category	**Very Small**	**Small**	**Medium**	**Large**	**Very Large**
Control	4.24	8.68	17.79	36.46	74.71
Display	4.72	6.35	8.55	11.50	15.46
File	3.30	6.23	11.74	22.14	41.74
Logic	6.41	12.42	24.06	46.60	90.27
Print	3.38	5.86	10.15	17.59	30.49
Text	4.40	8.51	16.47	31.88	61.71

31422 05 pp. 069-084 r3ah.ps 2/9/05 5:21 PM Page 78

In making a size estimate, first produce a conceptual design and identify its parts, and then decide on the functional category for each part and how many items it is likely to contain. Finally, determine whether each part falls into the very small, small, medium, large, or very large size range. From the relative-size table, you can then estimate the size of each part and calculate total estimated program size. There are various ways to produce the relative-size table, but the next section describes a relatively simple and reasonably accurate approach.

5.5 Producing the Relative-Size Table

The PROBE estimating method divides historical size data into categories that represent your kind of work. If, for example, you know the sizes of all the text-handling parts you previously developed, you can judge the likely size of a new text part. In my original PSP research work, I developed a total of 13 Pascal text objects, as listed in Table 5.2. Because of wide variation in the numbers of items per part, I normalized the data by the number of items and then sorted the data by LOC per item. This normalization results in the data shown in Table 5.2.

TABLE 5.2 PASCAL TEXT LOC PER ITEM

Object Name	Items	Object LOC	LOC per Item	Range
each-char	3	18	6.000	VS
string_read	3	18	6.000	
single_character	3	25	8.333	
each_line	3	31	10.333	S
single_char	3	37	12.333	
string_builder	5	82	16.400	
string_manager	4	82	20.500	M
list_clump	4	87	21.750	
list_clip	4	89	22.250	
string_decrementer	10	230	23.000	L
Char	3	85	28.333	
Character	3	87	29.000	
Converter	10	558	55.800	VL

The smallest number of LOC per item in Table 5.2 is now 6 and the largest is 55.8. Although this is nearly a 10-to-1 spread, it is somewhat more useful than the 30-to-1 spread without normalization. The value of knowing this range is clear when you consider the extreme case. Say, for example, you knew that the size range for text parts fell between 3 LOC and 3,000 LOC. This information would not be much help in picking the size of a new text part. Size ranges are thus most helpful if they are reasonably narrow.

The next step is to divide the size data into very small, small, medium, large, and very large size ranges. This provides an intuitive structure for judging part size. Some PSP and TSP support tools can generate the relative-size table for you, but you can easily make your own simple divisions by picking the largest and smallest values as the VL and VS sizes. Then pick the middle value for medium and the lower and upper quartile values as the S and L values, as shown in Table 5.2. If you wish to use a more mathematical method for selecting these values, the method I used to produce the values in Table 5.1 is given in Box 5.1.

BOX 5.1 PRODUCING A RELATIVE-SIZE TABLE

1. Divide the part sizes by the number of items in each part to give size per item.
2. Divide these item size data into functional categories that each have at least 6 to 8 members (calculation, text, and data, for example).
3. For each size value, take the natural logarithm of the size to give $\ln(x_i)$.
4. Calculate the average of these n logarithmic values: $\ln(x_i)_{avg}$.
5. Calculate the variance of the logarithmic values from their average:

$$\text{Var} = \frac{1}{n-1} \sum_{i=1}^{n} \left(\ln(x_\text{i}) - \ln(x_\text{i})_{avg}\right)^2$$

6. Calculate the standard deviation: $\sigma = \sqrt{\text{Var}}$.
7. Calculate the logarithmic ranges: $\ln(\text{VS}) = \ln(x_i)_{avg} - 2\sigma$, $\ln(\text{S}) = \ln(x_i)_{avg} - \sigma$, $\ln(\text{M}) = \ln(x_i)_{avg}$, $\ln(\text{L}) = \ln(x_i)_{avg} + \sigma$, $\ln(\text{VL}) = \ln(x_i)_{avg} + 2\sigma$.
8. Calculate size ranges: $\text{VS} = e^{\ln(VS)}$, $\text{S} = e^{\ln(S)}$, $\text{M} = e^{\ln(M)}$, $\text{L} = e^{\ln(L)}$, $\text{VL} = e^{\ln(VL)}$.

5.6 Estimating Considerations

Estimating is a skill that you can develop and improve. The following paragraphs describe techniques you can use to improve your personal estimating accuracy. These methods involve reducing estimating error, controlling estimating bias, and handling the common problem of having to estimate with limited knowledge of the product to be built.

Improving Estimating Accuracy

Estimating is tricky business; it is part analysis, part design, part projection, and part calculated risk. Even when you use effective estimating methods and produce sound conceptual designs, you will still make mistakes. This is normal. If it happens more than about one-third of the time, however, reexamine your practices. Calculate the errors between the actual and planned times and sizes to see whether you can identify some of the causes of the variations.

Even if you continue making fairly large estimating errors, you can improve estimating accuracy by tracking and analyzing your estimates and, where you can, subdividing the estimates into parts. For example, suppose you have historical data showing that the standard deviation (or error) of your size estimates is 25%. If you estimate a large job as a single 10,000-LOC unit, the likely error would be 25%, or 2,500 LOC. This would mean that, about 65% of the time, the actual size for this program would fall between 7,500 and 12,500 LOC, or 10,000 LOC plus or minus one standard deviation. Next, suppose that before you estimated this 10,000-LOC program, you subdivided it into 100 parts and had each part independently estimated by 100 different estimators. Because the average of these estimates would be about 100 LOC, the size of the combined estimate would be 100 times the average, or 10,000 LOC.

Now, assuming that all of these 100 estimators had the same 25% estimating error, each estimate would have a standard deviation of 25 LOC. To find the likely error of the combined estimates, you would combine their variances to get the variance of the total estimate. Because the variance is the square of the standard deviation, you would then take the square root of the variance to get the standard deviation of the total estimate. The standard deviation of each estimate is 25 LOC, so the variance is 625, giving 62,500 as the variance of the total estimate. The standard deviation of the total is then the square root of this, or 250 LOC. This means that about 65% of the time, the combined estimate would have an expected error of 250 LOC, or one-tenth the standard deviation of your original 10,000-LOC estimate. Making the estimate in 100 parts reduced the expected error by ten times.

Dividing an estimate into parts can help you to better understand the planned product and make more accurate judgments about its size. However, you won't get the statistical advantages of multiple estimates unless you use independent estimators. If you make all of the estimates yourself, you should still expect a 25% error for the total estimate.

Estimating Bias

Estimating errors are of two general kinds. First, estimating accuracy will fluctuate by some average amount (the standard deviation). Second, the estimates may have a bias. For example, you might tend to estimate 40% too low. Using the preceding example, for this 10,000-LOC program, your estimates would fluctuate around 6,000 LOC, not 10,000 LOC. Next, with the 100 estimators, if each estimated 40% too low, the nominal 100-LOC programs would be estimated with a bias of 40 LOC and a standard deviation of 25 LOC. Statistically speaking, the average of the sum of these estimates is the sum of their averages, so the combined estimate would then be 100 times the average, or 6,000 LOC. That is, the bias of 40 LOC would affect every estimate, and the resulting estimate for the total program would be around 6,000 LOC.

Again, the combined error of the independent estimates would be 250 LOC, meaning that about 65% of the time, the actual size would be expected to fall between 5,750 and 6,250 LOC. That is, it would be within about plus or minus 250 LOC of the biased average estimate of 6,000 LOC. Although this estimate looks even worse than the original 10,000 estimate, it actually is a much better estimate. In other words, the bias is likely to be a more consistent error that can be measured and at least partially compensated for, whereas the variance is not. If you were to gather data on many such estimates and calculate their biases, and if the estimating process were reasonably stable, you could make an accurate bias adjustment.

Estimate bias depends on the data you use, your estimating methods, and your knowledge of the application. It also depends on how much design work you have done, your degree of optimism about the job, and your estimating ability. The amount of an estimator's bias will fluctuate over time, and it can be radically influenced by such factors as management pressure, personal commitment, or the degree of error in the prior estimate.

To manage estimating bias, use an estimating method that relies on historical data. Then, consistently use average values in your estimating. If, for example, you believe that a new programming project has several very difficult functions, estimate the job in several parts and treat these difficult functions as relatively large. For the overall job, however, try to have as many small parts as very large ones so that the total estimate will come out to be about equal to your historical average. If you do this, individual estimates may be high or low, but your estimates will be correct on average. You will then have eliminated estimating bias.

An important objective of the PSP is to improve your ability to make estimates. Some developers' estimating errors swing quite widely between frequent large underestimates and large overestimates. As they learn to use historical size data and to track their estimating experience, however, their estimating ability gradually improves. Even the most experienced estimators, however, occasionally make significant errors. By measuring and analyzing your estimates, you will learn how to make better estimates, and how to compensate for your estimating biases. For example, if you always estimate too low, you will soon recognize this trend and begin to reduce your bias. Over time, then, your estimates will tend to fluctuate around zero.

The PROBE method, described in Chapter 6, uses the linear-regression method to achieve better estimates. The basic philosophy behind PROBE is to provide a procedure and the data for making progressively more accurate estimates. The defined procedure helps to produce more consistent results, and the data from each estimate will help you to gradually reduce the standard deviation and bias of the estimates. Furthermore, by gathering and using your own estimating data, your estimating accuracy will improve.

Incomplete Knowledge Bias

In considering estimating accuracy, there is another important source of error. When you estimate the size of a new program before you have designed it, you will be uncertain about precisely how big it will be. The only way to completely compensate for this uncertainty would be to complete the design and implementation before making the estimate. Of course, this is impractical when you need an estimate and a plan before making a commitment. In this situation, your best alternative is to gather all of the available historical data, use an estimating method such as PROBE that compensates for incomplete knowledge bias, and utilize the judgment of any available experts to help you make the best possible estimate.

Although the lack of product knowledge will increase estimating variance, its most important impact is on estimating bias. Because bias is a natural consequence of early estimates, it is important that you learn as much about it as possible. One approach for large projects is to reestimate the program's size at several points during development. Assuming that you make all of the estimates yourself, they should become increasingly accurate as development progresses. Over time, as you gather data on more projects, you will learn how much error to expect at each phase. To properly use these data, you would then use different regression parameters for each development phase.

On a TSP project, you make new size and resource estimates after completing the requirements work, again after completing much of the design work, and again after coding. If you kept a record of the phase for each estimate, you could compare the estimates with the actual results during the project postmortem. This would enable you to understand and at least partially compensate for the incomplete knowledge bias.

Overcompensation

If you consistently make underestimates or overestimates, examine your data and estimating practices. If the problems are principally with size estimating, you may not be using appropriate historical data, or you could be making too gross a conceptual design. Often, you can substantially improve your estimates by making a more complete conceptual design. Don't overdo it, however, as you should not complete the design before making the estimate. The objective is to learn how to make accurate estimates before doing the work. Conversely, if the problem is principally with the time calculations, examine the variation among seemingly similar tasks. Try to understand the factors that cause some tasks to take longer than others, and figure out how to plan for these factors. Remember that if you estimate everything as an "average" project, on average you will be right. Good methods and personal data produce balanced plans; experience and skill gradually reduce planning errors.

If you alternately underestimate and overestimate, try to stabilize your estimating process. However, don't try to adjust each estimate to compensate for the error in the previous one. Such intuitive adjustments are almost impossible to control. The danger in adjusting your intuition is that you could lose your intuitive sense by trying to treat it analytically. Make your adjustments in the data and estimating methods.

If your estimating accuracy continues to oscillate, seek help from a PSP instructor, a TSP coach, or a teammate. Ask them to review your estimate and to raise issues and questions. By explaining your estimates, you will better see how to improve them. Always make your own estimates, but do not hesitate to have others review and comment on them.

Selecting an Abstraction Level

If you pick a reasonably detailed abstraction level for your estimates, you will have more data points. The smaller the abstraction, the more actual versus estimated data you will get from each project. Conversely, if you pick too small an abstraction level, you could have trouble visualizing the small proxies. For example, if the typical parts for your programs have items that average about 10 LOC and there are 5 items per part, then the parts will average about 50 LOC each. When you later use the PROBE method to estimate a 50,000-LOC system, you will then have to identify, categorize, and estimate the sizes of about 1,000 parts. This could be a substantial job.

Large Systems

For systems much larger than about 5,000 LOC, it could be a major job to reduce the conceptual design to low-level parts. TSP teams typically handle this problem

31422 05 pp. 069-084 r3ah.ps 2/9/05 5:21 PM Page 84

by dividing systems into 10 to 20 or so major parts and estimating their sizes as a team. Generally, the combined experience of all team members provides enough historical data to enable TSP teams to make pretty good estimates, even before they have firm requirements.

Although such high-level abstractions are generally hard to categorize, once you have used PROBE on several projects, you will have learned how to use historical data to make size estimates, and you will have an intuitive feel for the sizes of even fairly large components and subsystems. Then, after completing some high-level design work, you will know enough to make much more detailed estimates.

Unprecedented Products

The PROBE method assumes that you can produce a conceptual design fairly quickly and that you can subdivide this design into familiar parts or components. Although this is not generally a problem for products of types that you have previously built, it is more difficult for new and unprecedented products. An *unprecedented product* is one whose size, complexity, performance, function, or some other important characteristic is so different from anything you have previously developed that you have no benchmarks to guide your planning or development work. The best solution in these cases is to resist making firm estimates for unprecedented products until you have completed a feasibility study and built some prototypes. This experience should provide the knowledge to refine the conceptual design to a level where you have historical data. If you can't do this, then you are no longer making an estimate, you are making a guess.

5.7 Summary

In project planning, estimates are generally required before you can begin development. At this early stage, you may generally understand the requirements but know little else about the product. Therefore, you will need a proxy to relate product size to the required product functions. The criteria for a good proxy are that its size should be closely related to the effort needed to develop the product and that it can be automatically counted. It should also be easy to visualize at the beginning of the job and be sensitive to language, style, and design differences that might affect the development effort.

In using a proxy, divide your data into ranges that correspond to the sizes of the proxy items in your database. With the PSP, the practice is to divide the proxy items into five size ranges: very large (VL), large (L), medium (M), small (S), and very small (VS).