Wrapped around these three steps is a more general step, a meta-step:

4. Provide estimates in ranges and periodically refine the ranges to provide increasing precision as the project progresses.

The following sections describe each of these steps in detail.

What Do People Usually Mean by "Estimate"?

"[The common definition of estimate is] *'An estimate is the most optimistic prediction that has a non-zero probability of coming true.'*

"Accepting this definition leads irrevocably toward a method called what's-the-earliest-date-by-which-you-can't-prove-you-won't-be-finished estimating."

—Tom DeMarco

8.3 Size Estimation

You can estimate the size of a project in any of several ways:

- Use an algorithmic approach, such as function points, that estimates program size from program features. (I'll summarize this practice shortly.)
- Use size-estimation software that estimates program size from your description of program features (screens, dialogs, files, database tables, and so on).
- If you have already worked on a similar project and know its size, estimate each major piece of the new system as a percentage of the size of a similar piece of the old system. Estimate the total size of the new system by adding up the estimated sizes of each of the pieces.

CROSS-REFERENCE
For more on measuring projects, see Chapter 26, "Measurement."

Most software programs and some of the algorithmic approaches require that you calibrate the estimation practice to your environment before you use it. Accurate measurement of historical projects is a key to long-term success in using any kind of estimation. (See the sidebar on the next page.)

Function-Point Estimation

CROSS-REFERENCE
Rules for counting function points get much more detailed than what's presented here. For details, refer to the "Further Reading" section at the end of this chapter.

A function point is a synthetic measure of program size that is often used in a project's early stages (Albrecht 1979). Function points are easier to determine from a requirements specification than lines of code are, and they provide a more accurate measure of program size. Many different methods exist for counting function points; the one I describe here is closest to the "1984 IBM Method," which is the basis of IBM's and the International Function Point User Group's (IFPUG's) current practice (Jones 1991).

Program Size

As it's used in this chapter, "size" refers in a very general way to the total scope of a program. It includes the breadth and depth of the feature set as well as the program's difficulty and complexity.

Early in the project, the most accurate way to think of size is often in terms of function points. Sometimes it's useful to think of size in comparative terms—for example, "Umpty-Fratz version 2 will have about 30 percent more functionality than Umpty-Fratz version 1" or "Umpty-Fratz version 2 should be about three-quarters as big as Foo-Bar version 4."

As you work your way through a project from requirements analysis to implementation and test, your notion of size changes, typically shifting from number of function points at requirements time to number of classes or modules at design time to number of lines of code at implementation and test time.

CROSS-REFERENCE For details on the relationship between function points and lines of code, see Chapter 31, "Rapid-Development Languages (RDLs).

The number of function points in a program is based on the number and complexity of each of the following items:

- *Inputs*—Screens, forms, dialog boxes, controls, or messages through which an end-user or other program adds, deletes, or changes a program's data. This includes any input that has a unique format or unique processing logic.
- *Outputs*—Screens, reports, graphs, or messages that the program generates for use by an end-user or other program. This includes any output that has a different format or requires a different processing logic than other output types.
- *Inquiries*—Input/output combinations in which an input results in an immediate, simple output. The term originated in the database world and refers to a direct search for specific data, usually using a single key. In modern GUI applications, the line between inquiries and outputs is blurry; generally, however, queries retrieve data directly from a database and provide only rudimentary formatting, whereas outputs can process, combine, or summarize complex data and can be highly formatted.
- *Logical internal files*—Major logical groups of end-user data or control information that are completely controlled by the program. A logical file might consist of a single flat file or a single table in a relational database.
- *External interface files*—Files controlled by other programs with which the program being counted interacts. This includes each major logical group of data or control information that enters or leaves the program.

The terminology in the function-point approach is fairly database oriented. The basic approach works well for all kinds of software, but you will have to fine-tune it for your own environment if you're not building database-intensive systems.

As Table 8-2 suggests, to compute the number of function points in a program, you take the number of low-complexity inputs in the program and multiply by 3, the number of low-complexity outputs and multiply by 4, and so on. The sum of those numbers gives you the "unadjusted function-point total."

Table 8-2. Function-Point Multipliers

	Function Points		
Program Characteristic	**Low Complexity**	**Medium Complexity**	**High Complexity**
Number of inputs	× 3	× 4	× 6
Number of outputs	× 4	× 5	× 7
Inquiries	× 3	× 4	× 6
Logical internal files	× 7	× 10	× 15
External interface files	× 5	× 7	× 10

Source: Adapted from *Applied Software Measurement* (Jones 1991).

You then compute an "influence multiplier" based on the influence that 14 factors have on the program, factors that include data communications, on-line data entry, processing complexity, and ease of installation. The influence multiplier ranges from 0.65 to 1.35.

When you multiply the unadjusted total by the influence multiplier, you get a total function-point count. Table 8-3 provides an example of how you would come up with that. The specific number of inputs, outputs, inquiries, logical internal files, and external interface files shown in the table are arbitrary. They and the influence multiplier were chosen solely for purposes of illustration.

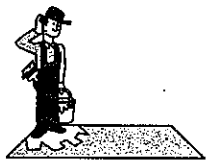
The program illustrated in Table 8-3 works out to a size of 350 function points. If you had come up with that number, you could compare it to the sizes of previous projects and their schedules, and you could estimate a schedule from that. You could also use Jones's First-Order Estimation Practice, described later in this chapter.

Table 8-3. Example of Computing the Number of Function Points

	Function Points		
Program Characteristic	**Low Complexity**	**Medium Complexity**	**High Complexity**
Number of inputs	6 × 3 = 18	2 × 4 = 8	3 × 6 = 18
Number of outputs	7 × 4 = 28	7 × 5 = 35	0 × 7 = 0
Inquiries	0 × 3 = 0	2 × 4 = 8	4 × 6 = 24
Logical internal files	5 × 7 = 35	2 × 10 = 20	3 × 15 = 45
External interface files	9 × 5 = 45	0 × 7 = 0	2 × 10 = 20
Unadjusted function-point total			304
Influence multiplier			1.15
Adjusted function-point total			350

Estimation Tips

Here are some general guidelines for making size estimates.



CLASSIC MISTAKE

Avoid off-the-cuff estimates. Developers are sometimes trapped by off-the-cuff estimates. Your boss asks, "How long would it take to implement print preview on the Giga-Tron?" You say, "I don't know. I think it might take about a week. I'll check into it." You go off to your desk, look at the design and code for the program you were asked about, notice a few things you'd forgotten when you talked to your manager, add up the changes, and decide that it would take about five weeks. You hurry over to your manager's office to update your first estimate, but the manager is in a meeting. Later that day, you catch up with your manager, and before you can open your mouth, your manager says, "Since it seemed like a small project, I went ahead and asked for approval for the print-preview function at the budget meeting this afternoon. The rest of the budget committee was excited about the new feature and can't wait to see it next week. Can you start working on it today?"

What just happened here? Was it bad management? Bad development? Poor communication? It was probably a bit of all three, and I've found that the safest policy is simply not to give an off-the-cuff estimate. You never know how far it will travel, and it's futile to try to put conditions on it; people have enough trouble trying to remember the estimate itself, and it seems as if they never remember the conditions you put on it.