A number of methods have been used to estimate software costs. They include

1. Algorithmic Models. These methods provide one or more algorithms which produce a software cost estimate as a function of a number of variables which are considered to be the major cost drivers. COCOMO is an example of an algorithmic model.

2. Expert Judgment. This method involves consulting one or more experts, perhaps with the aid of an expert-consensus mechanism such as the Delphi technique.

3. Analogy. This method involves reasoning by analogy with one or more completed projects to relate their actual costs to an estimate of the cost of a similar new project.

4. Parkinson. A Parkinson principle ("Work expands to fill the available volume") is invoked to equate the cost estimate to the available resources.

5. Price to Win. The cost estimate developed by this method is equated to the
price believed necessary to win the job (or the schedule believed necessary to be first in the market with a new product, etc.).

6. **Top-Down.** An overall cost estimate for the project is derived from global properties of the software product. The total cost is then split up among the various components.

7. **Bottom-Up.** Each component of the software job is separately estimated, and the results aggregated to produce an estimate for the overall job.

The strengths and weaknesses of each of these techniques are discussed in the sections below. The chapter ends with a set of conclusions regarding the use of these techniques.

### 22.1 ALGORITHMIC MODELS

Algorithmic models provide one or more mathematical algorithms which produce a software cost estimate as a function of a number of variables considered to be the major cost drivers. The most common forms of algorithms to be used for software cost estimation are

- Linear models
- Multiplicative models
- Analytic models
- Tabular models
- Composite models

Each of these types is discussed in turn, followed by a summary of the general strengths and weaknesses of algorithmic models.*

#### Linear Models

A linear cost estimating model has the form

\[
\text{Effort} = a_0 + a_1x_1 + \ldots + a_nx_n
\]

where \(x_1, \ldots, x_n\) are the cost driver variables, and \(a_0, \ldots, a_n\) are a set of coefficients chosen to provide the best fit to a set of observed data points. The development cost is generally obtained by multiplying the effort quantity by a constant cost for labor.

This is the type of model used in the large, pioneering software cost-estimation study performed by System Development Corporation in the mid-1960s [Nelson, 1966]. This study investigated over 100 candidate cost drivers, using 169 project data points, a larger sample than any other set of data points available today. The best linear model obtained from the data was a 13-variable model with a mean estimate of 40 man-months and a standard deviation of 62 man-months; not a very accurate

* A discussion of individual cost models is given in Section 29.7.
predictor. The best conclusion we can draw from this result is that there are too many nonlinear interactions in software development for a linear model to work very well.

**Multiplicative Models**

A multiplicative cost-estimating model has the form

\[
\text{Effort} = a_0 a_1^{x_1} a_2^{x_2} \cdots a_n^{x_n}
\]

where \( x_1, \ldots, x_n \) are again the cost driver variables, and \( a_0, \ldots, a_n \) are a set of coefficients chosen to best fit the observational data.

This is the type of model used in the IBM Walston–Felix analysis [Walston–Felix, 1977], in which the cost driver variables were constrained to the values \(-1, 0, \text{ and } 1\); and in the Doty model [Herd and others, 1977], in which the cost driver variables can assume only the values \( 0 \) and \( 1 \).

The multiplicative form of the model appears to work reasonably well if the variables chosen are reasonably independent (otherwise, one has problems with double counting of costs and interaction effects). The constraint that variables assume only values such as \(-1, 0, \text{ and } 1\) makes for somewhat unstable models; the cost estimates can change only in large steps. For example, the Doty estimate is multiplied by \( 1.83^1 = 1.83 \) if there is concurrent development of computer hardware, and by \( 1.83^0 = 1.0 \) if not; there are no intermediate multipliers.

**Analytic Models**

An analytic model takes the more general mathematical form

\[
\text{Effort} = f(x_1, \ldots, x_n)
\]

where \( x_1, \ldots, x_n \) are again the cost driver variables, and \( f \) is some mathematical function other than linear or multiplicative.

For example, the Halstead model [Halstead, 1977] takes the form

\[
\text{Effort} = \frac{\eta_1 N_2 N \log_2 \eta}{2S \eta_2}
\]

where \( \eta_1 \) is the number of distinct operators in the program; \( \eta_2 \) is the number of distinct operands; \( \eta = \eta_1 + \eta_2 \); \( N_2 \) is the total usage of all operands in program; \( N \) is the total usage of all operators and operands; and \( S = 18 \), approximately.

As another example, the [Putnam, 1978] model takes the form

\[
S_t = C_k K^{1/3} t_d^{1/4}
\]

where \( S_t \) is the software product size, \( C_k \) is a constant, \( K \) is the development effort in man-years, and \( t_d \) is the development time in years.
To date, the analytic models which have been developed contain only a small number of variables. Thus they are insensitive to a number of factors (for example, hardware constraints) which are often critical determinants of software cost.

**Tabular Models**

Tabular models contain a number of tables which relate the values of cost driver variables either to portions of the software development effort, or to multipliers used to adjust the effort estimate. The Aron model [Aron, 1969] consists of a simple $3 \times 3$ table relating development effort to project duration and project difficulty. The Wolverton model [Wolverton, 1974] estimates development effort as a tabular function of type of software, difficulty, and novelty. The Boeing model [Black and others, 1977] estimates a basic productivity rate as a tabular function of the type of software, and modifies the productivity rate by multipliers obtained as tabular functions of other cost drivers such as language, project size, novelty, etc.

Tabular models are generally easy to understand and implement, and are also easy to modify based on new cost driver insights. They may run into some difficulties based on the number of cost driver variables used in the tables. If only a small number of variables is used, the model will be insensitive to some important cost drivers. If a large number of variables is used, the model will have an even larger number of table values to calibrate, requiring a large data base for thorough model calibration and validation.

**Composite Models**

Composite models incorporate a combination of linear, multiplicative, analytic, and tabular functions to estimate software effort as a function of cost driver variables. Two commercially available software cost models, the RCA PRICE S model [Freeman-Park, 1979] and the Putnam SLIM model [Putnam-Fitzsimmons, 1979] are evidently composite models, although most of their internal details have not been published (some of the details of the SLIM model have been described in [Putnam, 1978]). The TR W SCEP model [Boehm-Wolverton, 1978] and the COCOMO model are also composite models.

Composite models have the advantage of using the most appropriate functional form for each component of the cost estimate. Their main difficulties are that they are more complicated to learn and to use by hand (this is the main reason that COCOMO provides simpler Basic and Intermediate versions as well as a Detailed version) and that they require more data and effort to calibrate and validate.

**General Strengths and Weaknesses of Algorithmic Models**

Compared to other estimation methods, algorithmic models have a number of strengths. They are objective, and not influenced by such factors as a desire to win, desire to please, or distaste for the project. They are repeatable; you can ask them the same question a week later and get the same answer. They are efficient and
able to support a family of estimates or a sensitivity analysis. And they are objectively calibrated to previous experience.

On the other hand, algorithmic models have several weaknesses. Since they are calibrated to previous projects, it is always an open question to what extent these projects are representative of future projects using new techniques, using new computer system architectures, dealing with new application areas, etc. They are unable to deal with exceptional conditions, particularly exceptional personnel, exceptional project teamwork, or exceptional matches (or mismatches) between the project personnel and the job to be done. And, like any other model, there is no way the model can compensate for poor sizing inputs and inaccurate cost driver ratings.

22.2 EXPERT JUDGMENT

Expert judgment techniques involve consulting with one or more experts, who use their experience and understanding of the proposed project to arrive at an estimate of its cost. The strengths and weaknesses of these methods are highly complementary to the strengths and weaknesses of algorithmic models.

On the strong side, an expert's judgment is able to factor in the differences between past project experiences and the new techniques, architectures, or applications involved in the future project. The expert can also factor in exceptional personnel characteristics and interactions, or other unique project considerations.

On the weak side, expert judgment is no better than the expertise and objectivity of the estimator, who may be biased, optimistic, pessimistic, or unfamiliar with key aspects of the project. It is difficult to strike a balance between the quick response expert estimate (timely, efficient, but hard to calibrate and rationalize), and the thorough, well-documented group-consensus estimate (soundly based and analyzable, but highly time consuming—and difficult to do all over again the following week when the specifications have changed somewhat).

Group Consensus Techniques: Delphi

Because of the many possible causes of bias in individual experts (optimist, pessimist, desire to win, desire to please, political), it is preferable to obtain estimates from more than one expert.

If estimates are obtained from a number of experts, there are a number of ways to combine them into a single estimate. One is simply to compute the mean or median estimate of all the individual estimates. This method is quick, but subject to adverse bias by one or two extreme estimates.

Another method is to hold a group meeting for as long as is necessary to get the experts to converge on, or at least agree to, a single estimate. This method has the advantage of filtering out uninformed estimates in general, but it has two main drawbacks. One is that group members may be overly influenced by the more glib and assertive members. The other is that group members may be overly influenced by figures of authority or political considerations.

One technique that has been used to avoid the drawbacks of the group meeting
is the Delphi technique [Helmer, 1966]. This technique was originated at The Rand Corporation in 1948 as a means of predicting future occurrences (hence the name, which comes from the location of the ancient Greek oracle), and has since been used as an expert-consensus method in various other applications such as corporate planning and cost estimation. Table 22–1 shows the steps involved in the standard Delphi technique. It has been used in various software cost-estimating activities, including estimation of factors influencing software costs [Scott–Simmons, 1974].

<table>
<thead>
<tr>
<th>TABLE 22–1 Standard Delphi Technique for Cost Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Coordinator presents each expert with a specification and a form upon which to record estimates.</td>
</tr>
<tr>
<td>2. Experts fill out forms anonymously. They may ask questions of the coordinator, but should not discuss the situation with each other.</td>
</tr>
<tr>
<td>3. Coordinator prepares a summary of the experts’ responses on a form requesting another iteration of the experts’ estimate, and the rationale behind the estimate (see Fig. 22–1).</td>
</tr>
<tr>
<td>4. Experts fill out forms, again anonymously, and the process is iterated for as many rounds as appropriate.</td>
</tr>
<tr>
<td>5. No group discussion is to take place during the entire process.</td>
</tr>
</tbody>
</table>

DELPHI COST ESTIMATION ITERATION FORM

PROJECT: Operating System

DATE: 6/21/83

HERE IS THE RANGE OF ESTIMATES FROM THE 1st ROUND

MAN MONTHS

Your estimate

Median estimate

36 MM

PLEASE ENTER YOUR ESTIMATE FOR THE NEXT ROUND.

PLEASE EXPLAIN ANY RATIONALE BEHIND YOUR ESTIMATE:

This looks like a standard process control operating system. The development team has had a lot of experience with such systems, and should have no trouble with this one.

FIGURE 22–1 Typical Delphi iteration form

A Delphi/Group Meeting Software Cost-Estimation Experiment

In 1970, an experiment was performed at The Rand Corporation to determine the relative merits of Delphi and group meeting techniques for estimating software costs [Farquhar, 1970]. Four groups were given the same software specification (for a large Air Force information system), which had taken 489 man-months to develop. Two groups used the standard Delphi technique, and two groups used a standard
half-day group meeting to arrive at a joint estimate. The Delphi groups did achieve an impressive convergence of some initially diverse estimates, but their results were considerably less accurate than the group meeting results as shown below:

<table>
<thead>
<tr>
<th></th>
<th>Delphi</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment A</td>
<td>217</td>
<td>485</td>
</tr>
<tr>
<td>Experiment B</td>
<td>1090</td>
<td>656</td>
</tr>
<tr>
<td>Actual</td>
<td>489</td>
<td></td>
</tr>
</tbody>
</table>

The extremely accurate estimate produced by the group meeting in Experiment A is an example of a group doing all the wrong things and still coming out extremely well (perhaps by coincidence). The group was dominated by a fairly assertive individual who proposed a Parkinsonian (see Section 22.4) solution: "Since there were 20 people available, they were all used full-time, and they must have finished in two years, because projects either finish in two years or they don't finish at all." This led the group to a very close estimate of 485 man-months in this instance, but subsequent attempts to apply such a method have been quite inaccurate.

**The Wideband Delphi Technique**

In reviewing the results of this experiment, Farquhar and I concluded that the written feedback in the standard Delphi technique did not provide a sufficiently broad communications bandwidth for the participants to exchange the volume of information necessary to calibrate their estimates with those of the other participants. This led to the formulation of an alternative method, which we called the Wideband Delphi technique, and which is summarized in Table 22-2.

**TABLE 22-2  Wideband Delphi Technique**

1. Coordinator presents each expert with a specification and an estimation form.
2. Coordinator calls a group meeting in which the experts discuss estimation issues with the coordinator and each other.
3. Experts fill out forms anonymously.
4. Coordinator prepares and distributes a summary of the estimates on an iteration form (similar to Fig. 22-1, but excluding the written rationale).
5. Coordinator calls a group meeting, specifically focusing on having the experts discuss points where their estimates varied widely.
6. Experts fill out forms, again anonymously, and Steps 4 to 6 are iterated for as many rounds as appropriate.

The Wideband Delphi technique has subsequently been used in a number of studies and cost estimation activities, for example, [Boehm and others, 1974]. It has been highly successful in combining the free discussion advantages of the group meeting technique and the advantages of anonymous estimation of the standard Delphi technique.
22.3 ESTIMATION BY ANALOGY

Estimation by analogy involves reasoning by analogy with one or more completed projects to relate their actual costs to an estimate of the cost of a similar new project. It is equivalent to the similarities and differences estimating technique discussed in [Wolverton, 1974].

As an example, one might say, "This environmental impact report generator for Oregon is similar to the one we developed for Florida last year for $1,200,000. The Oregon system has about 30% more types of reports than the Florida system had, so we'll add $360,000 to cover them. On the other hand, we'll be using many of the same people, so we can reduce the estimate by about 20%, or $240,000, to account for the time we spent getting up the learning curve on the Florida project. Also, we can save probably another 20% by reusing some of the low level report generation modules and most of the pollution model software, for another reduction of $240,000. Thus, our cost will probably be around $1,200K + $360K - $240K - $240K = $1,080K."

Estimating by analogy can be done either at the total project level or at a subsystem level. The total project level has the advantage that all components of the system cost will be considered (such as including the costs of integrating the subsystems), while the subsystem level has the advantage of providing a more detailed assessment of the similarities and differences between the new project and the completed projects.

The main strength of estimation by analogy is that the estimate is based on actual experience on a project. This experience can be studied to determine specific differences from the new project, and their likely cost impact. The main weakness of estimation by analogy is that it is not clear to what degree the previous project is actually representative of the constraints, techniques, personnel, and functions to be performed by the software on the new project.

22.4 PARKINSONIAN ESTIMATION

Parkinson's Law [Parkinson, 1957] says, "Work expands to fill the available volume." A Parkinsonian estimate takes the form

This flight control software must fit on a 65,536-word machine; therefore, its size will be roughly 65,000 words. It must be done in 18 months, and there are 10 people available to work on it, so the job will take roughly 180 man-months.

In some cases, a Parkinsonian estimate has turned out to be remarkably accurate. These have generally been cases in which the estimate left a good deal of extra time and money to continue adding marginally useful "bells and whistles" to the software until the budget ran out, at which point the software was declared complete. Even in these cases, it is not clear that this final embellishment phase made the best use of the people involved on the project.
There have been other cases in which the Parkinsonian estimate has been grossly inaccurate. An example is the flight control software estimate above; by the time the project was finished, it had added another 65,536-word on-board computer to accommodate the 127,000 words of software actually developed, and it required a total of 32 months and 550 man-months.

Parkinsonian estimation is not recommended. Besides not being particularly accurate, it tends to reinforce poor software development practice.

22.5 PRICE-TO-WIN ESTIMATING

Here are some examples of price-to-win estimating:

*I know the cost model estimated $2 million for this job, and none of our experts believe we can do it for less than a million and a half. But I also know that the customer has only $1 million budgeted for this software contract, so that's what we're gonna bid. Now go and fix up the cost estimate and make it look credible.*

*We absolutely have to announce this product at the National Computer Conference next June, and here it is September already. That means we've got 9 months to get the software ready.*

The price-to-win technique has won a large number of software contracts for a large number of software companies. Almost all of them are out of business today. The inevitable result is that the money or schedule runs out before the job is done, everybody gets mad at each other, a lot of compromises are made about the software to be delivered, and a lot of programmers work long hours just trying to keep the job from becoming a complete disaster.

The main reason that the price-to-win technique continues to be used is because the technology of software cost estimation has not provided powerful enough techniques to enable software customers or software developers to convincingly differentiate between a legitimate estimate and a price-to-win estimate. One of the primary objectives of the COCOMO model is to begin to provide a way for people to make these differentiations. It is possible to make the COCOMO model give you a lower cost estimate, but only by changing some objectively defined cost driver rating, whose validity can be checked by someone other than the estimator.

22.6 TOP-DOWN ESTIMATING

In top-down estimating, an overall cost estimate for the project is derived from the global properties of the software product. The total cost is then split up among the various components.

Top-down (and bottom-up) estimating can be done in conjunction with any of the methods discussed above. The examples given for analogy, Parkinsonian, and price-to-win estimating are also examples of top-down estimating.
The major advantage of top-down estimating is its system level focus. To the extent that the estimate is based on previous experience on entire completed projects, it will not miss the costs of system level functions such as integration, users’ manuals, configuration management, etc.

The major disadvantages of top-down estimating are that it often does not identify difficult low level technical problems that are likely to escalate costs; that it sometimes misses components of the software to be developed; that it provides no detailed basis for cost justification and iteration; and that it is less stable than a multicomponent estimate, in which estimation errors in the components have a chance to balance out [Wolverton, 1974].

22.7 BOTTOM-UP ESTIMATING

In the usual bottom-up estimate, the cost of each software component is estimated by an individual, often the person who will be responsible for developing the component. These costs are then summed to arrive at an estimated cost for the overall product.

Bottom-up estimating is complementary to top-down estimating, in that its weaknesses tend to be top-down’s strengths, and vice versa. Thus, the bottom-up estimate tends to cover just the costs associated with developing individual components, and to overlook many of the system level costs (integration, configuration management, quality assurance, project management) associated with software development. Bottom-up estimates are often underestimated as a result.

A bottom-up estimate also requires more effort than does a top-down estimate, but in many respects this is an advantage. In particular, having each part of the job costed by the person responsible for its success will be very helpful in two main ways.

1. Each estimate will be based on a more detailed understanding of the job to be done.
2. Each estimate will be backed up by the personal commitment of the individual responsible for the job.

Further, a bottom-up estimate tends to be more stable, in that the estimation errors in the various components have a chance to balance out.

The most effective way to ensure that system level costs are included in a bottom-up estimate is to organize the software job into a work breakdown structure (WBS) which includes not only the product hierarchy of software product components but also the activity hierarchy of project jobs to be done (see Chapter 4, Figs. 4-6a and b). This ensures that the costs of such activities as integration and configuration management are included. However, unless their estimates can be delayed until the estimates for the product components are established, they may be inaccurate; for example, it is difficult to estimate integration costs without a good idea of the size and nature of the components to be integrated.
The Task-Unit Approach to Software Cost Estimation

The traditional approach to software cost estimation, and the one used most frequently for bottom-up estimation, is the task-unit approach. In this approach, the job of developing a software component is broken up into task units. The effort required for each task unit is estimated, generally by the component developer, and the resulting estimates summed to produce the overall effort estimate for the software component. An example is given in Fig. 22-2.

The major advantages of the task-unit approach are those of bottom-up estimating: it promotes a deeper understanding of the job to be done, and it allows the individual developers to plan their own jobs, ensuring their personal commitment to the resulting estimate, and giving them more control over their own project destinies. In addition, the task estimates provide a sound basis for overall project planning and control, as will be discussed in Chapter 32.

The main difficulties with the task-unit approach are those of overlooking system

FIGURE 22-2  Sample task unit planning sheet

<table>
<thead>
<tr>
<th>COMPONENT</th>
<th>Task Unit</th>
<th>DEVELOPER</th>
<th>W Ward</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase</td>
<td>Man-days</td>
<td>Totals</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plans and requirements</td>
<td>Component requirements</td>
<td>5</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Development plan</td>
<td>1</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>Product design</td>
<td>Product design</td>
<td>6</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Draft users' manual</td>
<td>3</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Test plan</td>
<td>1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Detailed design</td>
<td>Detailed PDL</td>
<td>4</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Data definitions</td>
<td>4</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Test data and procedures</td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Full users' manual</td>
<td>2</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Code and unit test</td>
<td>Code</td>
<td>6</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Unit test results</td>
<td>10</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>Integration and test</td>
<td>As-built documentation</td>
<td>4</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Integration support</td>
<td>5</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Grand Total</td>
<td></td>
<td>53</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
level costs, and those of overlooking two significant additional sources of software cost. These are

1. **Incidental project activities**, which may add another 30 to 50% to the amount of effort spent on producing the items enumerated on the Task Planning Sheet in Fig. 22–2. Figure 22–3 shows the distribution of effort by activity for the two small application software projects discussed in [Boehm, 1980]; it shows that such activities as reading, reviewing, meeting, and fixing consumed roughly 40% of the development effort for both projects.

![Figure 22-3](image)

**FIGURE 22-3** What does a software project do? Distribution of project effort by activity.

2. **Incidental nonproject activities**, which may add still another 30 to 50% to the amount of effort devoted to all the project activities. Figure 22–4 shows the results of a Bell Laboratories time and motion study of 70 programmers [Bairdain, 1964], indicating that roughly 30% of the programmer's workday is devoted to nonproject activities: training, personal business, nonproject communication, etc.

These difficulties should not be considered as fundamental drawbacks to the task-unit approach, but more as considerations to be covered in reviewing task-unit estimates for completeness. On balance, the task-unit method is an extremely valuable one, and the most appropriate approach to software cost estimation for very small (less than 2000 DSI) software projects.
22.8 SUMMARY COMPARISON OF METHODS

Table 22.3 summarizes the relative strengths and weaknesses of the software cost-estimation methods discussed in this chapter. The main conclusions that we can draw are

- None of the alternatives is better than the others from all aspects.
- The Parkinson and price-to-win methods are unacceptable and do not produce sound cost estimates.
- The strengths and weaknesses of the other techniques are complementary (particularly the algorithmic model versus expert judgment and top-down versus bottom-up)
### TABLE 22-3  Strengths and Weaknesses of Software Cost-Estimation Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algonthmic model</td>
<td>• Objective, repeatable, analyzable formula</td>
<td>• Subjective inputs</td>
</tr>
<tr>
<td></td>
<td>• Efficient, good for sensitivity analysis</td>
<td>• Assessment of exceptional circumstances</td>
</tr>
<tr>
<td></td>
<td>• Objectively calibrated to experience</td>
<td>• Calibrated to past, not future</td>
</tr>
<tr>
<td>Expert judgment</td>
<td>• Assessment of representativeness, interactions, exceptional circumstances</td>
<td>• No better than participants</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Biases, incomplete recall</td>
</tr>
<tr>
<td>Analogy</td>
<td>• Based on representative experience</td>
<td>• Representativeness of experience</td>
</tr>
<tr>
<td>Parkinson</td>
<td>• Correlates with some experience</td>
<td>• Reinforces poor practice</td>
</tr>
<tr>
<td>Price to win</td>
<td>• Often gets the contract</td>
<td>• Generally produces large overruns</td>
</tr>
<tr>
<td>Top-down</td>
<td>• System level focus</td>
<td>• Less detailed basis</td>
</tr>
<tr>
<td></td>
<td>• Efficient</td>
<td>• Less stable</td>
</tr>
<tr>
<td>Bottom-up</td>
<td>• More detailed basis</td>
<td>• May overlook system level costs</td>
</tr>
<tr>
<td></td>
<td>• More stable</td>
<td>• Requires more effort</td>
</tr>
<tr>
<td></td>
<td>• Fosters individual commitment</td>
<td></td>
</tr>
</tbody>
</table>

Thus, as we discussed in Sections 21.5 and 21.6, it is important to use a combination of techniques, and to compare and iterate the estimates obtained from each. The particular combination we choose will depend on our cost-estimation objectives (for example, more top-down for rough early estimates; more bottom-up for detailed planning estimates). In general, however, an effective combination is the following:

- A top-down estimate using the judgment of more than one expert, using analogy estimation where a comparable previous project is available.
- Bottom-up estimation using an algorithmic model, with inputs and component-level estimates provided by future performers.
- Comparison and iteration of both estimates.

The component level Intermediate COCOMO model presented in Chapters 8 and 9 can be used to support bottom-up algorithmic model estimates, but often a more detailed model with more hierarchical levels will be more effective. The next chapter presents the Detailed COCOMO model, which can provide this level of support.

### 22.9 QUESTIONS

22.1. Define the seven methods of software cost estimation discussed in this chapter.

22.2. Perform a Delphi exercise to estimate the size of the operating system on your main computer, and compare the results with its actual size. Did the range of estimates in the successive Delphi rounds decrease? Did the median estimate converge more closely to the actual size?
22.3. Can you think of other possible software cost-estimation techniques which do not fall within the categories discussed in this chapter? What would be their relative strengths and weaknesses?

22.4. (Research Project) Perform a comparative study of the standard Delphi technique and the Wideband Delphi technique on a software cost-estimation activity.

22.5. (Research Project) Perform comparative studies of the relative effectiveness of different algorithmic models over a range of software cost-estimation situations.

22.6. Suppose you were the president of Query Systems, Inc., a commercial software house. Your company has prepared a proposal to develop a data base management system and an interactive query system for a hardware vendor with a new minicomputer. Your performers have given you an estimate of $1200K to do the job, and your marketing expert tells you that the price to win is $800K. What should you do?