First, Get The Front End Right

Lawrence H. Putnam

Michael Mah

Ware Myers

The development team settled down in its bullpen one fine Monday morning in April, ready to begin work on its new project. The lead developer said, "You folks start coding. I'll saunter over to executive row and see what they want."

Saunter? Instead he/she should rush over to his/her gassed-up time machine, set the destination control to the preceding January 1, and find out what the requirements are, whether there are critical risks that would make the project dubious, what the architecture will probably be, what the likely amount of work will be, and how much time and effort to budget for that. There is also the matter of getting in some design time before starting to code. Why, some people even claim that you should start test planning back somewhere in time-machine time!

Unless our hero/heroine is leading a very small project in a very well known area, he/she is due for a rude awakening as he/she saunters down this path of code-first, requirements-second. In real life software developers are not yet privileged to have access to the time machine that the higher reaches of our society appear to believe in. Software organizations have to develop software the hard way, that is, they have to plan *in advance* time for the early phases.

Now, it is possible that we may have exaggerated the plight of our hero. Usually, in that first meeting, there is a sense of "what they want." There are often project overview documents, but they are written in general prose, not in the more formal language in which requirements and specifications are eventually cast. Moreover, these initial memoranda are never complete. Our work-a-day developers still have a lot of information to dig out.

Still, before the developers and the client have pulled together much of this necessary knowledge about the project, the client likes to set a deadline. "We need this by next April Fools' day," they say. "That is our president's birthday." (There we go again! Just fooling around.)

Suppose that date is still 12 months off. The developers estimate that these fairly vague memos outline about 125 person-months of effort, based on their experience with several previous projects. They know they have only four people handy at the start, so maybe they can get in only four person-months the first month. That leaves 121 person-months and 11 months. They'll have to ramp up to 11 person-months per calendar month. The cost estimate will be 125 person-months times some labor rate, plus some more for other expenses.

It is the purpose of this article to consider the role that the core metrics--size, time, and effort--play in the early phases of a project. (The fourth core metric, defects, is supremely important, of course, but we will deal with that another day.) These early phases are known by various names, but they are often called the feasibility study and high-level design. The relationships of the core metrics and these two phases are extremely complex. Moreover, they feed back on each other. It will take some mental gymnastics to get the picture.

Estimation accuracy improves as knowledge increases

Whether a software organization is in the fix of our hero or whether it has some "memo" knowledge, it still doesn't know what acute risks the project faces. It doesn't really know how much work will be involved; it is just guessing. Consequently, it is in no position to predict with much accuracy the two core metrics--time (calendar months) and effort (person-months)--on which the project cost estimate is based. In contrast, advanced estimating systems depend upon what the *size* of the completed system will be. By size, we refer to a number such as source lines of code. To get to even an estimate of such a number, projects have to do some preliminary work.

Figure 1 shows that software size-estimation accuracy improves during the development phases. Early in the feasibility phase the spread in the estimate may be as great as several hundred percent of the final size. That spread narrows as the phases proceed and all concerned learn more. It is not until the system is ultimately released, however, that the software organization can pinpoint the actual size.

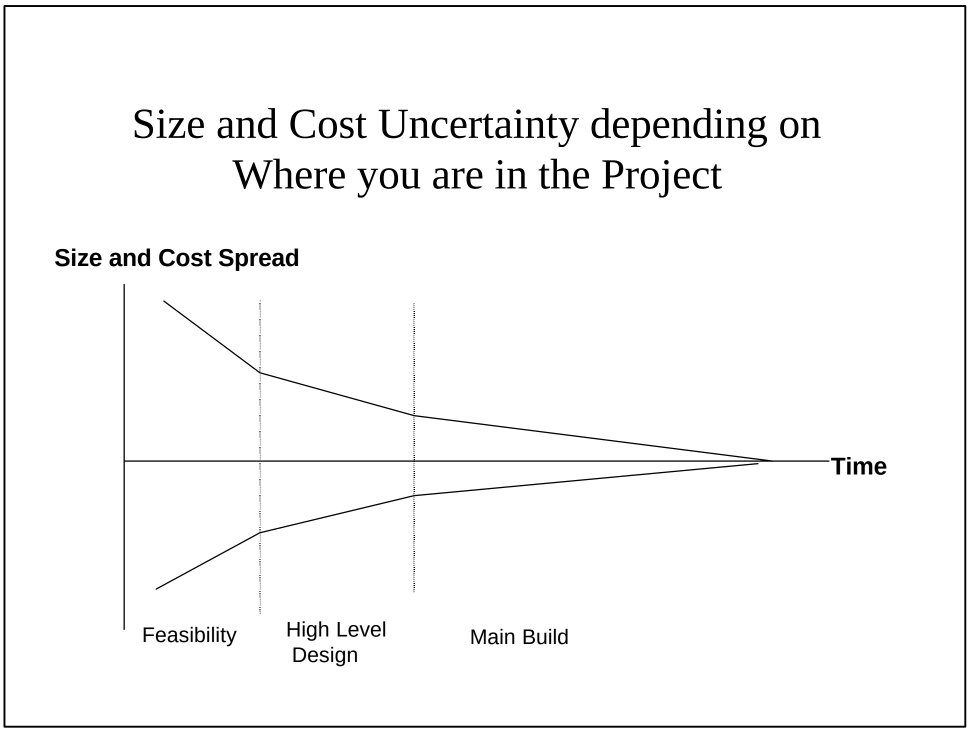
Size and Cost Uncertainty depending on Where you are in the Project
Size and Cost Spread
Time
Feasibility
High Level Design
Main Build

Figure 1. In the feasibility phase, when little is known about the system to be built, estimates of its size necessarily have a wide range. As work proceeds, the ability to estimate size improves.

This size-convergence curve depicts the nature of the problem that stakeholders face in planning a software development. In the beginning they would like to have some idea of how long development will take and how much it will cost, but they are stymied by the fact that they know little about the proposed system. Knowing little, they cannot estimate the system size very closely. Yet system size is a key value in the estimating tools with which sophisticated estimators calculate effort and development time. And those two core metrics, in turn, are the basis of the cost estimate.

Put this way, the need for a size estimate may seem somewhat obvious, yet the first parameter clients often set is the deadline. "We haven't figured out what it is, but we know we need it by October 1," they say. As an afterthought, they may add, "Oh yes, do it for one million dollars." Then a second afterthought, "Send your "boy" over next week to find out what we want."

Look back at Figure 1. Software development doesn't work that way. In fact, the software industry has a mountain of failed projects to prove it. To the contrary, if we take the size-convergence curve as gospel, we have three logical courses of action.

Wait. Defer making an estimate of size (and hence of schedule, effort, and cost) until you reach the point on the curve at which a reasonably firm estimate becomes possible.

Don't pinpoint. If you have to make an estimate before you reach the firm-estimate point on the curve, offer a range estimate, for example: 18 to 24 months schedule, most likely delivery in 21 months; $700,000 to $1,300,000 cost, most likely figure $1,100,000. Conceivably you and the client could agree to re-bid periodically, narrowing the bid range, as the project becomes better understood. In actual practice, re-bidding is not unusual, though usually not by prior agreement. Rather, as the end of the original schedule or funds approaches, the software organization and the client fight over extending the time and effort.

Expect changes. If you have to make a point estimate, make it clear that the information to support that point figure is not yet firm and that you will be back to negotiate changes in schedule and cost (effort) as you learn more.

We hear you saying, "Practical business people won't do these things. They want firm bids and deadlines." Of course they do. On the one hand, they have customers clamoring for delivery, loans coming due, and profits to make. But, on the other hand, they have the reality of the size-convergence curve staring at them. Irresistible force meets immovable object. Fine philosophical dilemma! Let's explore it. We'll start by looking at what happens during the early phases.

The early phases reduce uncertainty

The early phases are engaged in exploring risks and establishing high-level design (or architecture). From an estimating point of view, these efforts reduce uncertainty.

Resolve critical risks. A risk so serious as to imperil the completion of the project presents a high degree of uncertainty. The feasibility-phase team identifies risks of this magnitude early in the first phase. Of course, they have to bound the proposed system in order to know where to look for this critical risk. To set the scope of the system, they have to capture the core requirements and outline a possible architecture.

They need to carry requirements capture and architecture only far enough to identify the critical risks. If they find any, they must at least outline a way in which each one can be mitigated, perhaps not immediately, but during the early phases. For instance, there may be a technical solution; there may be a modification of the requirements that places the critical risk outside the scope of the system to be built; there may be a subcontractor with special experience in the field of the risk.

Establish feasibility. If the first-phase team finds a way to mitigate the critical risks, the system is feasible. The team may also have to consider other issues affecting the feasibility of undertaking the project, such as:

- Do the people available have the qualifications and experience the project calls for;
- Can the software organization establish effective relations with the client's people?

Some of the bid-no bid considerations may play a role in the feasibility decision.

Capture requirements. One of the factors affecting the size-convergence curve is the interaction between requirements and architecture. If the core requirements were firm at the outset, it would be possible to settle on an at least one workable architecture. All too often, the requirements change drastically as the developers and the stakeholders explore what is to be done. When this happens, a double jeopardy occurs. First, requirements *change* has a deleterious effect on team efficiency (its process productivity falls). Second, requirements *growth,* or scope creep, creates an inescapable pressure on the deadline and budget.

Initiate architecture. Once a core architecture has been set forth, it is possible to get a fairly good grasp of the functionality of the system. Functionality is expressed for estimating purposes in some kind of size metric. For example, the proposed system can probably be partitioned into 13 subsystems, nine of which are well known, two are components that can be reused, and two present some new problems. Whether we express the functionality in subsystems, function points, use cases, object classes, or requirements "shall" statements, all are convertible into size in source lines of code. "Size", ultimately expressed in SLOC and other units of measure, are the beginning inputs in advanced estimating methods.

However, requirements are usually not firm in the early phases. It may take quite a bit of analysis to move from the existing business process, or even a proposed business-engineered process, to the requirements that express the process. Moreover, it can turn out to be impractical to devise an architecture that implements all the desired requirements. In other words, exigencies in the architecture may reflect back into the requirements that the project can accommodate. Sometimes early-phase prototypes of interfaces or algorithms remind users of additional requirements.

In sum, the requirements grow, the architecture develops, and the size increases in response. Pressure on the deadline rises. The size estimate remains uncertain as system elements are gradually firmed up during the early phases, but the degree of uncertainty declines, as the size-convergence curve shows.

Explore significant risks. After the project team perceives a way to mitigate the critical risks, its next target is the significant risks. These are risks that the team is certain it can surmount, but at a cost in time and effort that is initially indeterminate. To reduce estimation uncertainty, the team carries its analysis of significant risks to the point of assuring itself that they can be resolved within the time and effort to be available in the main-build phase. Attaining this assurance may involve searching for appropriate algorithms and interfaces, building prototypes, or searching for reusable components.

Make the business case. During the early phases the project team and other parties concerned with the business side make the business case. Broadly speaking, there is no point to going ahead with a software project if it does not make business sense. The initial business case may merely ascertain that the project is feasible, that the effort and cost are within the resources available to the client, and that the product can reach internal application or the marketplace in time to pay off. For these purposes the broad range of the early size-convergence curve may be close enough.

Toward the end of the early phases, before the main build commences, management needs a closer reading of the business case. It is at this point that the software organization makes the estimate on which management bases the business bid. Just how firm that estimate has to be, just how narrow the uncertainty zone should be, that is the question that we address next.

How to risk-buffer an estimate

As we proceed through the initial phases, we gradually resolve many of the issues. The functionality and size estimate gets more accurate. That, in turn, leads to an improving estimate of the time, effort, and cost of the remainder of the development period. In fact, once we make a firm bid and receive a go ahead, we are in the main build.

One can conceive of carrying the preliminary phases up to the point at which we have firm requirements, an equally firm architecture, and all important risks resolved. Under those circumstances we could make a firm estimate. We could derive a bid that we would have a very good chance of achieving. However, the rigors of business competition lead to the necessity for software organizations to submit bids well before that point of adequate knowledge.

Also, business and government are accustomed to point bids. The various forms of range bids that we enumerated earlier are not acceptable. We have to make a point bid. Appreciating the reality of the convergence curve, can we buffer the schedule and effort that we bid to accommodate the remaining uncertainties?

Yes, we can. To do so, however, we have to grasp the value of the principles of probability, statistics, and simulation. We don't have to master the computational mechanics of these fields. Software tools can do that work. We merely have to understand the big picture.

Uncertainty drivers. Advanced estimating tools figure time and effort from three inputs: size, process productivity, and staffing rate. All three are uncertain, as diagrammed in Figure 2, but size is the principal villain.

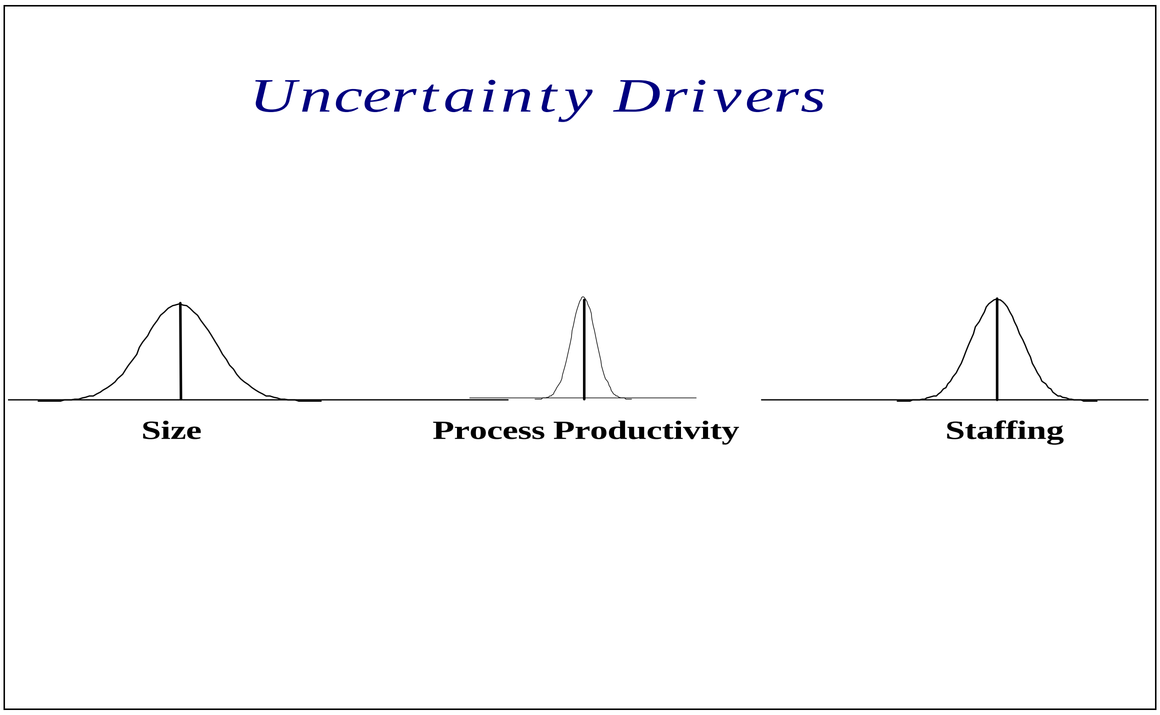

Figure 2. Size, process productivity, and staffing rate are sources of uncertainty and hence of risk in software estimating.

Advanced estimating methods derive process productivity, basically, in one of two ways. One is by calibration from completed projects, where time, effort, and size are known. (In this case, note that you have to have these core metrics. In other words, they are important to the business side of software development.) If you can derive this input by calibration from past projects, their value is quite certain. Their possible spread around the nominal value is small.

The second way is by evaluation by managers of the project organization's efficiency. Some of the advanced estimating methods use this way exclusively. Other methods resort to it when the organization lacks the core metrics on past projects. In that case its spread around the nominal value will be much larger and, consequently, contribute more uncertainty to the time and effort estimate.

Similarly, estimating methods approach staffing considerations from the same two angles--calibration from past metrics or evaluation.

Size, of course, must be estimated the hard way by capturing requirements, managing risks, and formulating the architecture. In the first phase the spread of size around the nominal value will be very large, as Figure 1 shows. As you learn more, this spread contracts.

If you have to make your bid early in the high-level design phase, the size spread is still substantial, as Figure 1 shows. If you can wait to the end of this phase, the spread is much less. Bidding usually takes place somewhere in this second phase. (The names of these points are different in various organizations.) That means that, at best, you are quite uncertain of the size of the system you are trying to bid.

It may not be entirely obvious, but the rather large uncertainty about the size of the proposed system and the lesser uncertainties about process productivity and staffing do carry through the advanced estimating methods to the estimates of schedule and effort. They will not be the precise numbers you would like to have underlying your bid. The answer to this dilemma is found in Monte Carlo simulation.

The simulation concept. Figure 3 presents five concepts and we will deal with them in turn. First, look at the single line extending from northwest to southeast, so to speak, labeled Size and PI (meaning Productivity Index, a number representing process productivity). This line means that effort and schedule are interlocked. If you increase the time allowed (within reasonable limits), the amount of effort required decreases. In any given project situation, the values of the effort and schedule lie somewhere on a line such as the one illustrated. That line, of course, lies somewhere on the (log log) effort-schedule plane, depending on the specifics of the project at hand.

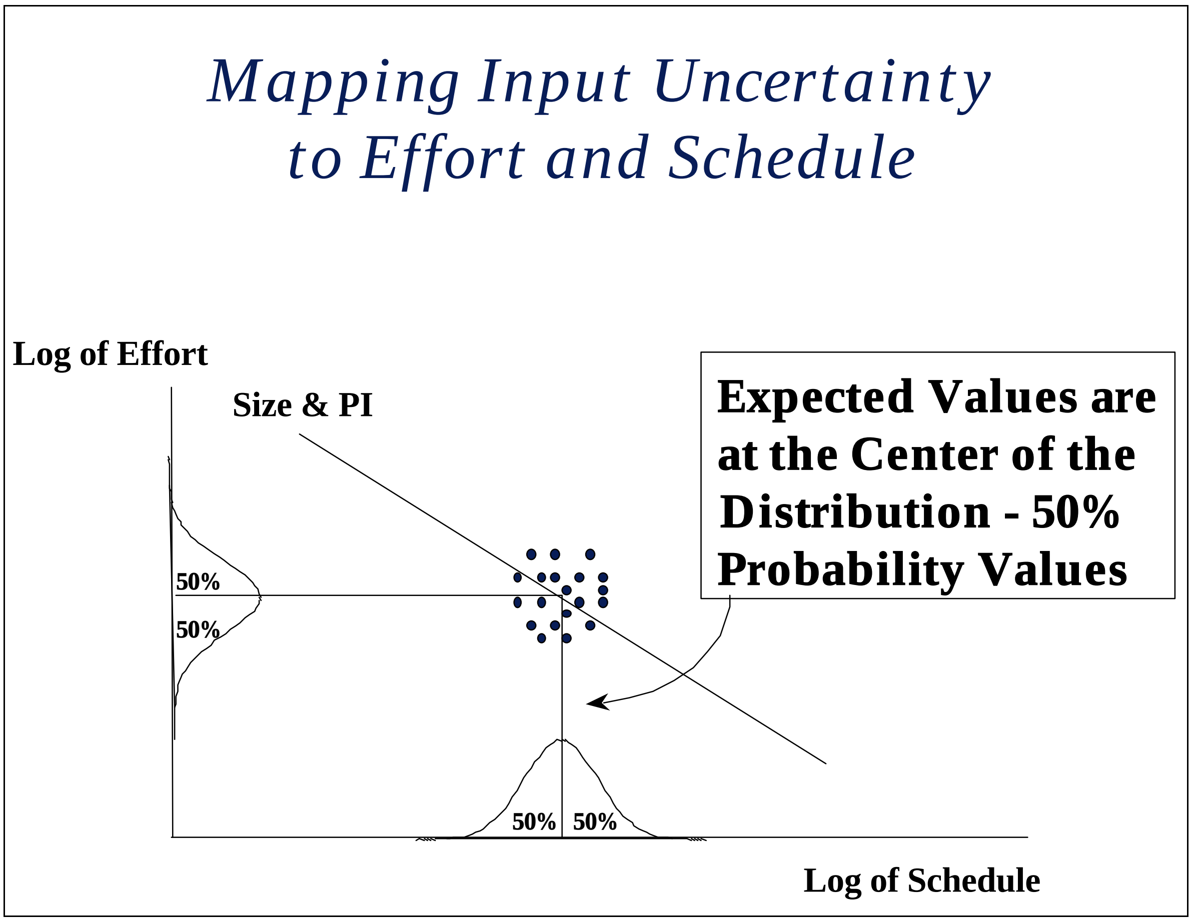
Mapping Input Uncertainty to Effort and Schedule
Log of Effort
Size & PI
Expected Values are at the Center of the Distribution - 50% Probability Values
50%
50%
50%
50%
Log of Schedule

Figure 3. Monte Carlo simulation carries out the operation diagrammed here.

Second, in the case of any given project, its particular line may drift up or down (vertical to the line itself) by reason of the uncertainties in the size, productivity, and staffing inputs.

Third, look at the circle of dots. The center of this circle is the proposed operating point of the project, that is, a given schedule and effort. We can vary the operating point up and down the line if we wish. If we solve the two equations a dozen times with various inputs within the uncertainty range, we get a dozen operating points, signified by the dots. With today's modern PCs, we actually run the solutions a thousand or more times, getting a thousand or more dots. The computer program varies the input frequencies according to the uncertainty curves of Figure 2. The output dots follow this frequency distribution, heavy toward the center of the circle of dots, light toward the circumference.

Fourth, look at the curve on the schedule axis. This curve is proportional to the frequency of the output dots. The most likely output is the one at the center, or peak, of the curve. Those far from the peak are less likely.

Fifth, look at the curve on the effort axis. The value at its peak is the most likely value of the effort, but many other values are possible.

This diagram visualizes the reality that the estimates of schedule and effort, given uncertain inputs, are themselves uncertain. Consequently, a cost estimate, based on these estimates of schedule and effort, will also vary. Management must necessarily base a point bid somewhere within the circle. At the center of the circle, or at the midpoint of the effort and schedule curves (on the axes), the probability is 50 percent that the project can be accomplished at those values.

If management lowers the bid by reducing the effort and time planned, the schedule-effort operating point moves over into the southwest quadrant of the circle of dots. The probability of successful completion falls below 50 percent. At the lowest value of an effort-schedule dot, unfortunately, the probability of successful completion falls to near zero. In other words, if management reflects its exuberance in a low bid, it is setting itself up for an almost certain project failure.

If management wishes to be more conservative, it can select an operating point in the northeast quadrant--allowing the project more schedule and more effort. The probability of successful completion of the project then goes up to 70 percent, 80 percent, or more.

Without the benefit of the probability-statistics-simulation analysis we have just summarized, management is often tempted to go with the low bid. Granted, it is important to win contracts. It is also important to complete projects successfully and make a little money while doing so. It is also important to stay in business. This type of analysis gives you the opportunity to gauge your chances.

If you have competitors for a project, remember, they are up against the same set of uncertainties that you are. If they understand the bidding situation, they, too, are going to bid a number that they have a good chance of making. If they don't understand the effect of these uncertainties--and many don't--let them bear the onus of failure.

If you are dealing with a government unit that believes it is mandated to go with the low bidder, you might try explaining that the laws often intend the lowest *qualified* bidder. A software organization that bids in ignorance of the odds is not exactly qualified. The government unit is storing up trouble for later. If you are dealing with private organizations or your own in-house management, try to explain the facts of uncertainty and how they work out.

Better yet, if a software organization responds to a request for proposal with a demonstrable history of outstanding cost and schedule performance on its past projects, that speaks strongly about its being the right organization for the job. This demonstrated history is all about the core metrics for past projects, which go to the issue of "Knowing, and Showing, Your Capability".

Estimating the early phases

So far we have talked about estimating the main build itself. But the phases before it also take time and utilize effort. Management often wants some idea of how much time and effort they will take.

Feasibility study. As soon as we can make a very rough estimate of the size of the proposed system, we can make an equally rough estimate of the time and effort of the main build. Then the rules of thumb for estimating the feasibility study are:

- Schedule: about one quarter of this main-build schedule.
- Effort: about five to 10 percent of the main-build effort.

These estimates are uncertain to the same extent that the main-build estimates are uncertain at this early point. They are also uncertain because the rules of thumb are rough.

High-level design. Similarly, we can estimate the time and effort of this phase with these rules of thumb:

- Schedule: about 30 to 35 percent of the estimated main-build schedule.
- Effort: about 20 percent of the estimated main-build effort.

The longer you can delay making this estimate, the less the uncertainty of the size estimate and the better the high-level design-phase estimates will be.

So, get the front end right

The software process consists of a series of overlapping and iterative phases. Developers can accomplish the tasks of each phase more successfully if management is in a position to provide the time and effort each phase properly takes. In particular, a software organization can estimate the time and effort the main build will take more accurately if the early-phase team can reach a good estimate of the system size. Reaching that good estimate, in turn, takes time and effort, for which management has to make allowance--itself another estimate.

All these estimates depend upon the core metrics--size, time, and effort. These core metrics are also characterized by the reality that they are uncertain, more so in the early phases. Consequently, the estimating methodology has to be able to cope with uncertainty. Fortunately, management can maneuver through the thickets of uncertainty by applying the principles of probability, statistics, and simulation.

No doubt you have detected a certain circularity in our argument. You need metrics to make good estimates. You need good estimates to control the software process. But the metrics values are uncertain, casting a shadow of doubt on the estimates. And without adequate allowances of time and effort, the software process itself limps.

Woe is us! Where to attack this circle? All around it, of course, but at the front end in particular.