Familiar Metric Management: Size

Hidden Rule #1: "We can't make sensible business decisions with such wide unknowns," upper managers say.

From Tom DeMarco, *Why Does Software Cost So Much?*, Dorset House Publishing, 1995, 237 pages.

If we knew how much "work" a software project involved, we would be well on the way to estimating its schedule and effort. Later, during the "work," schedule and effort are the two projections against which we principally run statistical controls. Effort over the time span of a project at a process productivity rate represents the "work" embodied in the resulting product.

Effort applied over a period of time is what the waterfall model of software development represented. Many estimating methods are based on that model. However, only occupants of organizational niches far removed from the software tar pit ever believed this model literally represented reality.

Since the 1970s we have seen many more models appear: prototyping, rapid development, spiral, iterative design, domain analysis (leading to reuse), and periodic release. In each case, some number of lines of code is created. This number is usually only a fraction of the total number of lines of code in the entire product. In any event the size of this fraction is the fundamental metric with which project estimating begins. We believe source lines of code to be the best available stand-in for project "work."

Even organizations that don't keep metrics generally have an idea of how many SLOC they are putting out. Naturally, SLOC being familiar, it is the measure we use in Familiar Metrics Management-Size.

In FMM-S then, the first step is to estimate the size of the software product—or portion of the product currently contemplated. "The future is unknowable," the Unknown Soldier (or some equally famous ancestor) is reputed to have said. Nevertheless, commercial software development depends upon pushing fearlessly into the unknowable. Maybe "fearlessly" is not quite the right attitude with which to penetrate this terra incognita. "Fearlessly" is the word that characterizes those who elbow into it without benefit of metrics.

Estimating team

Still, a size estimate does not grow on the nearest bush as we enter this jungle. It must be carefully built up on four fronts.

Facts. The first need is for the facts about the proposed software product. What is it going to do? At first, in the arm-waving stage, we might hazard a guess that it will be about as big as "Star Float," a project we worked on in our youth. Unfortunately, remembrance of our youth is a weak reed on which to hang an estimate. As we gather requirements, write specifications, and lay out functional design, we accumulate the facts that make a more accurate estimate possible.

Estimators. The second need is for people. Perhaps you hoped that the metrics crowd were hiding some magic wand from you. Sorry. People have to do it. But not just any people. In particular, not just people who occupy the more glittery rungs at the top of the brass ladder. Rather, the most experienced and knowledgeable people you have. At least two or three of them, so the perennial optimist can offset the sober pessimist.

Data. Even good people's judgment doesn't spring full-blown from their noble brows. It rests upon knowledge

that comes from data recorded from past projects. Your forward-looking estimates are more valid to the extent that you have kept and use the familiar metrics. Don't bury them in masses of musty project records. Put them in a computer database that your good estimators can access.

Time. Give these good people time to make a good estimate. Getting acquainted with the proposed project, accessing the historical project data, and working through an estimating process take time.

Estimating process

The estimating team does more than sit in a room together for a few days. It should follow an agreed-upon "process." With a process, the team can devote itself to the facts in the case, not to debating how to go about making an estimate. With a known process, the levels of management above the team can focus on the merits of the particular estimate, not quarrel with how the team went about its work.

When you are focused on "facts," you can work quietly. When you have not agreed on "method," you can rant loudly, if ineffectually. In estimating, the presence of a formal process focuses attention upon the facts in a repeatable manner. A formal process, in turn, rests upon three legs.

Familiar metrics. Both the team and the levels above them work to the same metrics, such as those familiar "management numbers" we have been describing. **Estimate ranges**. "The future is unknowable." Remember? And one aspect of it that is especially unknowable is the exact size of the product your project is going to produce 12 or 36 months into it.

1. We know the requirements are going to grow. They reflect the part of the world the system is to deal with and the world is always changing. Moreover, as the project proceeds, the users and the developers learn what the real needs are, forcing requirements growth, or, at least, change.
2. If the project is to extend over the product life of the hardware on which it is to run, the next generation of hardware may force code changes.
3. Similarly, the software methods, practices, tools, etc. that you employ will likely change. (You are trying to improve your process productivity, aren't you?)
4. If you are in a competitive arena (as who isn't), those old "debbils," your competitors, may toss you a few curves.
5. Finally, unless you are producing in your sleep the 17^{th} version of the system, you don't know exactly how you are going to solve all the problems lying ahead of you. In other words, you don't know precisely how many SLOC you will wind up with 12 or 36 months down the trail.

Moral: don't expect your estimating team to specify 56,278 source lines of code. Ask them rather to indicate a range:

- Low
- Most likely
- High

Ask them to set Low and High so that the chance of the eventual size lying within that range is 99 percent. (That happens to be the +\-3 standard-deviation range, handy knowledge to use a little later in estimating statistical risk.) The most likely size is not necessarily in the exact middle of the Low-High range.

The “expected” value of the size estimate, then, is the Low plus the High plus four times the Most Likely—all divided by six, or, Expected Size = [Low + 4(Most Likely) + High]/6.

This process weights the most likely judgment heavily. “Expected” value is what you expect, statistically, the ultimate size to be. The word “expected” carries with it a big disclaimer. It means there is a 50-percent probability that the actual size will be greater than this “expected” value, anywhere along the plus three-standard deviation range. Or a 50-percent chance that it will be less, along the minus three standard-deviation range, as visualized in Figure 1, shown below.

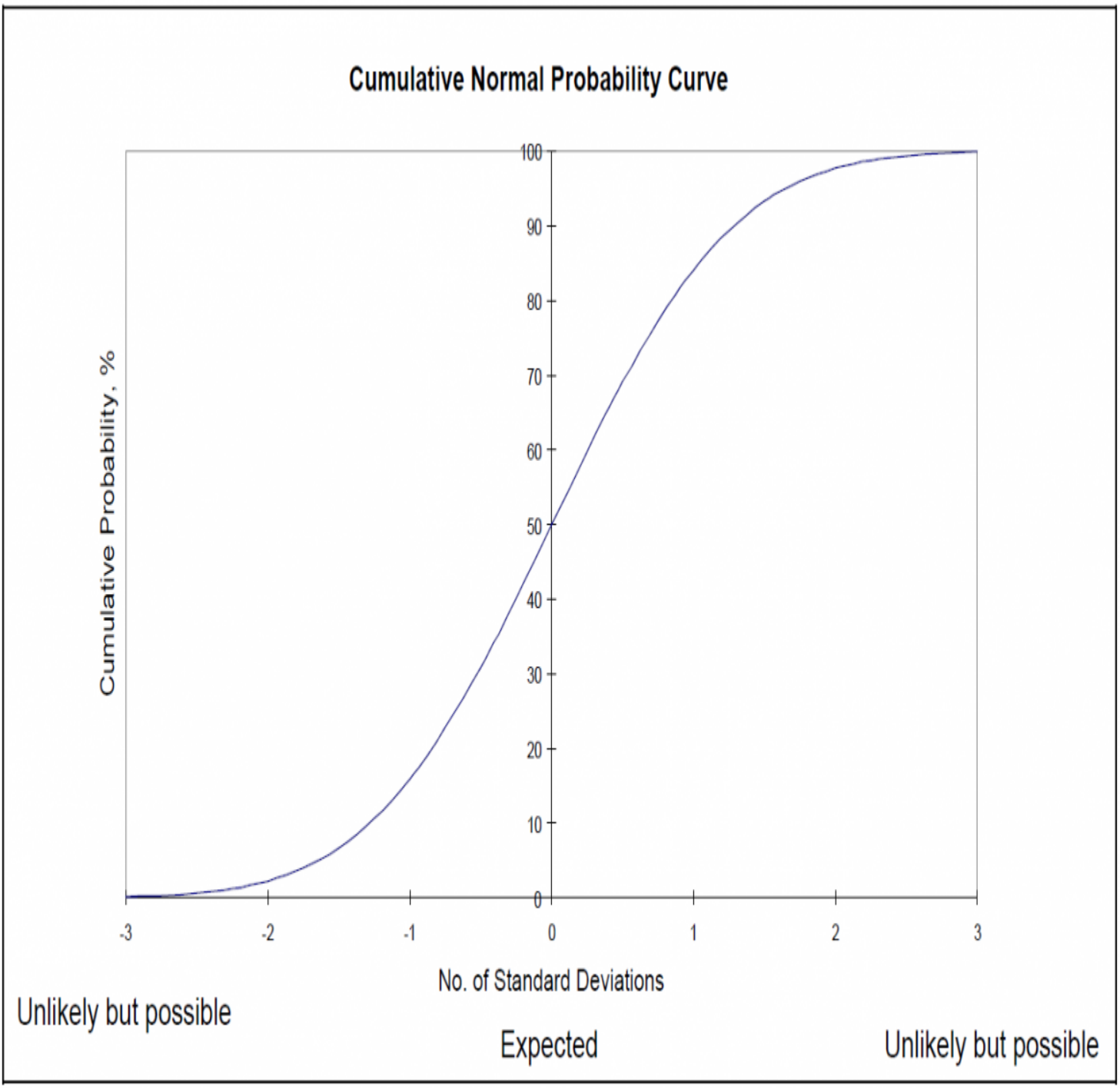

Figure 1. The familiar "bell-shaped" normal curve of statistical fame, drawn here in cumulative form, shows how the expected size estimate is likely to turn out.

The High minus the Low represents a six-standard-deviation range. Dividing it by six provides an estimate of one standard deviation, or, Standard Deviation = (High Low)/6.

Many things in life follow the normal statistical distribution. Software size estimating pretty much does, too. So, we don't wind up with an exact estimate, but we do have some statistical concepts to work with. For example, we can expect that about two third of the actual sizes of a series of projects will fall within plus and minus one standard deviation of the expected value, as shown in the figure. That does not give us an estimate

precise down to the nearest line of code, but it does reduce the size of the ballpark quite a bit.

For bidding or budgeting purposes you may have to submit an exact figure, but at least you know that the figure is really a range.

Accept risk. Risk is a reality in software development— no secret. Getting statistics into the estimating process gives you a means of assessing the degree of risk. In the present case, for example, bidding at the expected or 50-percent level, you might win or lose on any one project. Over a series of projects you could expect to break even.

You are not limited to bidding at the 50-percent level. There are statistical methods for bidding at any level you choose. If you consistently bid at the 70-percent level, you would expect to make a profit over a series of projects. If you consistently "buy in" at the 30-percent level, you can expect to lose over a series of projects.

We must bring the hidden rules, such as DeMarco's, "we can't make sensible business decisions with wide unknowns," out in the open. Otherwise, "they gull us into making the same mistakes over and over and over again," he says. With well constituted estimating teams, following an established process, accepting the risk involved, we can deprive the hidden rules of their power over software development.