Familiar Metric Management: The Software Cost Estimation Problem is Solved!

So stated Tom DeMarco at the European Software Control and Measurement Conference in 1995. [fn]Tom DeMarco, "Function and Disfunction," *Proc. European Software Control and Measurement Conference,* ESCOM Science Publishers B. V., Leiden, Netherlands, 1995.[/fn] It came to our attention in a recent article by Barbara Kitchenham and Stephen Linkman. who added: " DeMarco understood this as early as 1982." [fn]Barbara Kitchenham and Stephen Linkman, "Estimates, Uncertainty, and Risk," *IEEE Software,* May-June 1997, pp. 69-74.[/fn] That is when he published his book on the subject. [fn]Tom DeMarco, *Controlling Software Projects: Management, Measurement, and Estimation,* Yourdon Press, New York, 1982, 284 pp.[/fn] Barry Boehm had published his book on the subject the year before. [fn]Barry W. Boehm, [italics] Software Engineering Economics, Prentice Hall, Upper Saddle River, NJ. 1981, 767 pp.[/fn] We ourselves had published papers on the subject in 1978 and a IEEE Computer Society tutorial in 1980. [fn]Lawrence H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Transactions on Software Engineering,* July 1978, pp. 345-361[/fn][fn]Ware Myers, "A Statistical Approach to Scheduling Software Development," [italics] *Computer,* Dec. 1978, pp. 23-35.[/fn][fn]Lawrence H. Putnam, *Software Cost Estimating and Life-Cycle Control: Getting the Software Numbers,* IEEE Computer Society, 1980, 349 pp.[/fn] The tutorial republished 19 papers and hundreds more are extant.

Obviously, DeMarco meant that "the problem has been solved in principle." One has only to look around to see that it has not been solved in practice. In fact, Kitchenham and Linkman note a little farther on in their article: "A major problem remains: although software managers know what to do, they just don't do it."

We have a little trouble with the first half of this conclusion. In our experience, most software managers are not familiar with "what to do." After all, about three quarters of software managers work in SEI Level One organizations. In any case, the fraction that do have some grasp of "what to do" have trouble doing it. That is the topic we address in this column.

Why it is hard to estimate software development

Let us exclude from this discussion the very considerable body of software work that, essentially, is not estimated. That includes most of the maintenance work that is carried out on a level-of-effort basis. That is, some number of programmers is given the task of maintaining an organization's software. They work away at it. If they get too far behind, management adds a few more programmers to their number. No one makes much effort to estimate the jobs they do. Much new development work is carried on in the same way. If too many promising new projects pile up, management increases the level of effort.

That leaves the jobs that are estimated for us to consider. Some of the bigger ones get written up in the newspapers a few years later. Some big organization was going to automate all their flyswatters. That is, on a certain date in the spring, when the flies appear, all the flyswatters would come out of the closet and lay quietly on the desks, ready for use.. Well, lah di dah, came spring and the flyswatters stayed on the closet's top shelf; the program could not reach them! Substitute IRS, FAA, DoD, BoA, Y2K, or any other three-letter combination for flyswatters and you get the picture.

What characterizes these big jobs, besides being big.. Two things. One is that they are not very well defined to

begin with. The “requirements” are incomplete and even wrong in many respects. This requirements problem is not something that can be corrected by clean

living and right thinking. It is inherent in complex projects. The

customers/users/stakeholders don’t know what ought to be done beyond the knowledge that spring (and flies) will come. If they do, on occasion, spend a lot of time on requirements, as DoD has been known to do, they come up with documents of thousands of pages. Methods of dealing with that much information are not well developed.

The second characteristic of big jobs is that they entail risk. They are going to accomplish something big. The State of California, for instance, has a project to automate the tracing of fathers who are supposed to be paying child support. Its roots go back to 1980 when the Federal government urged the states to develop such systems. In 1988 the Family Support Act required the states to have tracking systems in place by 1995. California initiated its effort in 1991, estimating a cost of $99 million. Now the cost estimate has grown to $300 million and work is still under way. One difficulty is that the software has to work with legacy software in the 58 counties. That was a “risk,” since nobody knew at the start of the project what problems it would encounter in different counties. By the way, California’s experience is representative—only three states have completed their systems.

So far, two points: (1) Big projects start with a big vision, but few nuts and bolts that you can put a cost on. (2) Big projects entail “risks,” usually lots of them. Depending on what you count as a risk, there may be dozens of them, or even hundreds of them

Third point. In the intermediate reaches of the executive structure are “bottom-line” people who want a bid down to the nearest penny and the exact day of delivery. It may not have escaped your notice that there is a discrepancy here. “We don’t know in full what we are going to do, but we do know enough to see risks in it. Nevertheless, we promise to deliver it on July 31 two years from now for $29, 642,788.”

“We have found that risks (or at least the acknowledgment of them) diminishes the closer you get to the US Congress,” observed Marvin J. Carr of the Software Engineering Institute. [fn]Marvin J. Carr, “Risk Management May Not Be for Everyone,” *IEEE Software*, MayJune, 1997, pp. 21-24.[/fn] In general, placeholders in a bureaucracy are uncomfortable with risk. Solution: the vision proponents from the lower reaches pretend there are no risks or, if a few turn up, “our splendid team will vanquish them on schedule.”

Well, all this, and we haven’t even reached the problems of estimating per se. Even if we knew with reasonable exactitude what we were going to do (and, in software, we often don’t), we would have to know how long it took us and how much it cost us to do something like it the last time. There are two problems there. First, we have to have a record of last time (and, not so incidentally, we have to know where to lay our hands on it). Only the software organization can keep its own metrics. No one can do it for you.

Second, we have to be able to compare “last time” (or a dozen or two previous projects) to next time. Software projects are not easy to characterize for comparison purposes. Various schemes have been proposed for this purpose. For example, we have our process productivity factor, described in previous columns. Estimating methods based on the Cocomo concept assign numerical values to a score or so of cost drivers. In principle, by comparing the current assessment of these cost drivers to past assessments, estimators can make a workable comparison to past projects.

What we ought to do

If we are going to move software estimating from the realm of principle, where it has rested for the last two decades, into widespread practice, we have to move on five fronts:

1. Keep some metrics;
2. Accept the fact that risks exist;
3. Accept the reality that requirements develop over time;
4. Work toward getting clients to let you size the risks and nail down the requirements before expecting a firm bid;
5. Base estimates on probability, for example, the probability is 80 percent that we can complete this project for X dollars in Y months.

One more addendum. There are two basic ways of accommodating that last point. One is within the software organization itself. Above the level of the project, general management sets aside a fund, levied on the profits of “lucky” projects, to cover the losses of “unlucky” ones. The other is in the hands of the client organization. Knowing that software bids are uncertain, it establishes a contingency fund from which to fund run-on contracts to complete projects that came out on the wrong side of their probability estimate.