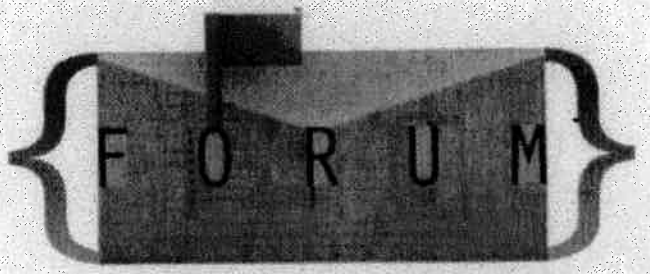

Planning for Software Project Success

I READ ROBERT GLASS'S "Practical Programmer" column ("Evolving a New Theory of Project Success," Nov. 1999, p. 17) and would like to add a practitioner's point of view.

First, I agree software projects need "More accurate and honest estimates. More understanding of the complexity of scope. More expert help." If we had all this, we'd be in great shape. But there are obstacles, often insurmountable, to each of these goals.

Regarding accurate estimates, it is difficult to estimate the project's effort fully when the scope is not fully known ahead of time. Yet it is almost impossible to know the full scope until the project is finished or you have previously done exactly the same work on exactly the same platform. The latter case, in my experience, is rare. Nevertheless, good programmers with a little experience can make rough estimates. However, when the estimates "seem" to be too high, either to management or to the client, the reaction is: "It can't possibly take that long. What could you do to take that long?" I have heard similar reactions from both internal management and from clients. The sad fact is that if, as in our case, we are doing custom work for outside clients, you don't win many jobs with high estimates. So projects proceed, with boss or client directly or indirectly mandating a schedule that is not based on the programmers' best guesses.

Related to the estimation issue is the understanding of the complexity. Aside from the point about not knowing you are there until you are, software development has evolved from the specification-first waterfall method to various iterative approaches. This means everybody presumably agrees all the details won't be known until a project begins, and revisions are made along the way, including a certain amount of "scope creep." The iterative approach works best for the end user but means the full complexity is unknown at the outset. (It also can cause problems from a sales point of view because it is difficult to convince clients to fund a project in small stages or to be open to changes in cost as the project evolves; or alternatively, a set price that in the end causes a loss of money.)

One conclusion then, from Glass's column, is that if scope creep is bad and understanding everything up front is good, then we should go back to the waterfall method of ironclad specifications as the first step. But hasn't history shown, especially from the user's point of view, that this is not the best approach?

Finally, we come to the matter of expertise. In my experience, on almost all projects, the programmer is doing something he or she has never done before, whether it involves hardware, software, or application details. Even if the programmer uses the same software development environment as before, chances are good that new (to the programmer) functions will be used, or known functions will be used in new ways. The programmer is rarely really and truly expert in the job at hand, if we take "expert" to mean a master who knows everything there is to know about the task. If our programmers are not experts, can they seek expert advice? Usually not. There may be no such person who knows everything about the task, or if there is, that person may be impossible to find. Or, if the advice is found, it is too expensive. Generally, good programmers ask around, send

PAUL WATSON

email, post queries on bulletin boards, and read articles and books in an attempt to learn more about how something works. But this is all time consuming, costly, and may not lead to a good answer anyway. Then there are the situations in which the programmer simply doesn't realize that expert advice at a given point could prevent trouble later. In practice then, I believe that while full expertise is certainly desirable, it is rarely achievable.

If Glass knows how we can get around these obstacles, especially in a competitive environment where we have to pitch project schedules and costs to skeptical clients, I'm listening.

RICHARD H. VEITH
Port Murray, NJ