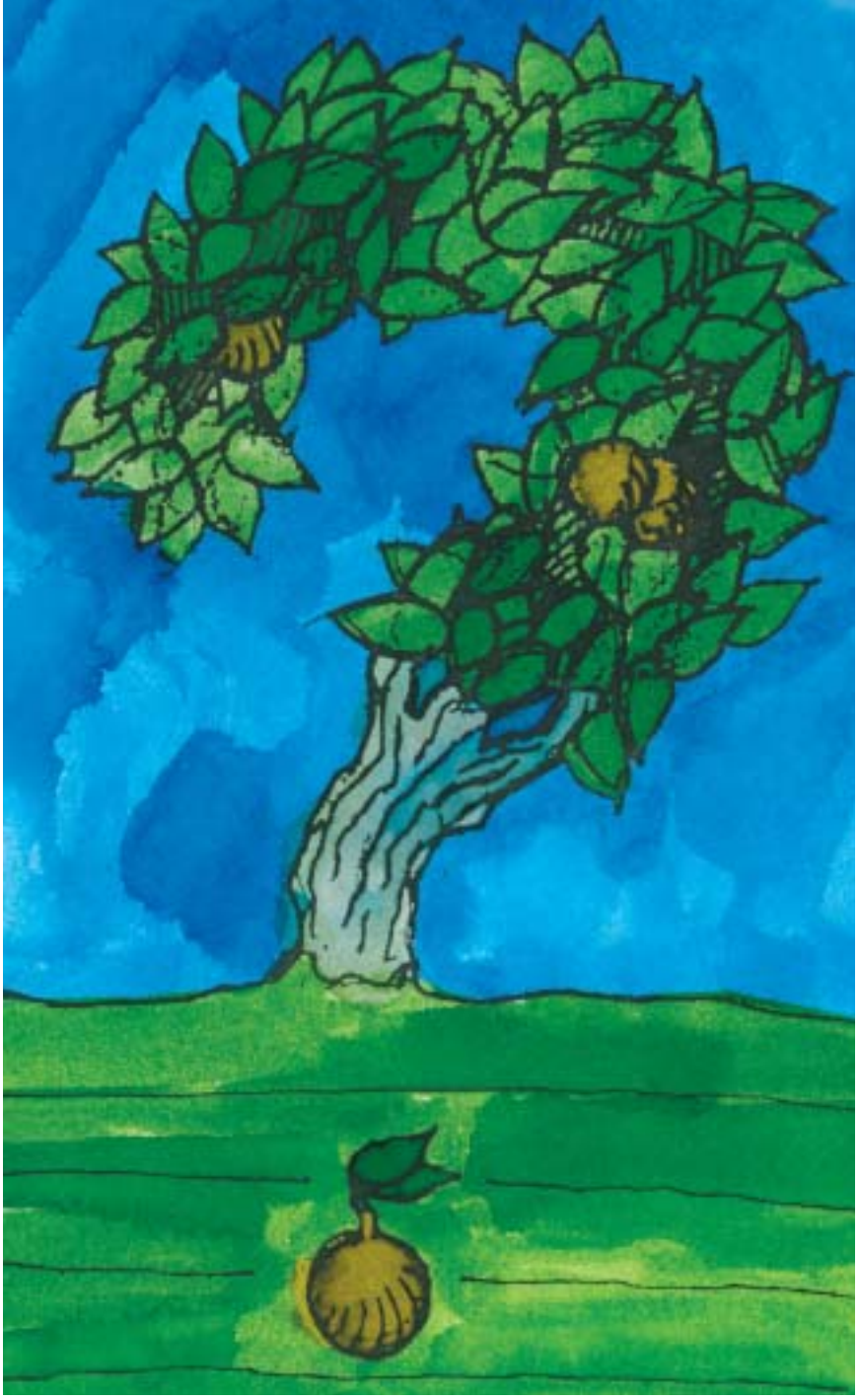

# REQUIREMENTS

## THE MORE THE BETTER?

BY TERRY LITTLE

For as long as I can remember, we in the Department of Defense have based our development programs on user requirements. The process is more or less sequential. The user develops the requirements, supposedly driven by the threat, and the developer structures a program to satisfy them.

Great process, right? Nope. It's a bankrupt process that has failed time after time.

In truth, most of the so-called "requirements" are really "desirements." They have little or nothing to do with the threat or real need, but everything to do with what the user or sponsor thinks may be possible. And what's the worst part? Much of what ends up in the requirements stem from what overly optimistic, fiscally naive program people—government or contractor—eager to "sell" the program have told the user or sponsor they can do.

**Being overly constrained by too many requirements with too little wiggle-room will invariably create problems.**

When the chickens come home to roost, as they always do, one can invariably trace many a program problem to having written unrealistic expectations into the requirements. It's a self-fulfilling prophecy.

How does one deal with this? Simple. Nothing ever, ever, becomes a requirement until two things happen: (1) there is a solid understanding and acceptance of the requirement's cost and schedule implications; and (2) knowledgeable technical people are so confident that the program can meet the requirement within the cost and schedule that they are willing to bet their jobs on it.

Yikes!! Does this mean that we never undertake high-risk projects? No. What it does mean is that when you undertake high-risk activities, you agree on an expectation or requirement that includes failure, or falling short, as a real possibility, and your cost and schedule reflect the risk. The other thing that it means is that you may have to start a project with some requirements open until after the work progresses to a point where the requirement meets the two criteria above.

The process is also flawed because there are usually too many requirements. Something about the engineering or designer mentality seems to demand hosts of requirements as an input to the technical process. Granted, it's more comforting to have someone else issue the requirements than it is to have to derive them. But, being overly constrained by too many requirements with too little wiggle-room will invariably create problems. In a perfect world, a sponsor's requirement would only be to obtain a certain capability with the detailed technical requirements derived from what's truly possible.

Linked to this issue of over-specification is the problem of prioritization of requirements. If everything is equally important, then nothing is important. In my estimation, a program or project should start with only a few key requirements. "Few" means not more than four or five, and "key" means that, if the program doesn't meet these requirements, it should be terminated.

Finally, within DoD cost is almost never a requirement, much less a key requirement, but it should be—always. Cost is a technical issue. We can only buy as much technical requirement as we can afford. It's true when we buy a car, and it's also true when we initiate a project.

My practice has been to put cost—which may be development, production, or both, depending on the program—into the technical performance specification. Why do this? I do it to make clear that everyone on the project, most especially the technical people, must own cost just as they do performance. Cost can't be just the concern of the project manager or the budget analyst. Everyone has to own it.

So what does all this mean? Simple. One sows the seeds of a project's success or failure during the requirements stage. With bad seeds the fruit will be bitter indeed. •

**TERRY LITTLE** is the Director of the Kinetic Energy Boost Office at the Missile Defense Agency. One of the most seasoned program managers in DoD, he is also a regular contributor to *ASK Magazine*.