Phillip G. Armour

Twenty Percent

Planning to fail on software projects.

In my March column, I described four projects I encountered at a company where management had signed up for what appeared to be only a 20% probability of completion within the committed schedule. Well, actually, management hadn't signed for such a low probability at all, because they really didn't know what the probability might be.

This is not unusual. Software projects are routinely committed at uncertain levels of probability in all kinds of companies. Sometimes even the concept of probability applied to software development receives a blank stare.

Disorganized Crime

I first encountered this in the early 1980s when, as a project manager, I presented an estimate to my boss. To say this gentleman was a tough manager was an understatement: he would not tolerate any thought or mention of failure. When he asked the all-too-common question:

> "When will this project be done?" I replied: "We have a 67% probability of not exceeding either budget or schedule by more than 20% if we aim for this date."

Bad move.

He scowled and repeated: "When will this project be done?" Probabilities and percents were so much mumbo-jumbo to him. He wanted two things: complete assurance that whatever schedule we picked, we would guarantee to deliver the system by that date; and that the schedule would be very short and would match or beat what he had already promised to his management and customers. These are contradictory conditions, of course, but I have heard from literally hundreds of software developers, engineers, and managers over the years that this can be the "as normal" state in the business of software.

Because he was both my boss and the software development equivalent of a leader of a crime syndicate, I promised a date that met the second criterion. But we did not make the date. Over the years at this and other companies I observed that this commitment process was not unusual and, when applied, was rarely successful. In fact, it became something of a game between the developers, the management, and the customer. Commitments would be made that pretty much nobody expected to be achieved; certainly the development staff did not. In talking with customers, it was

MICHAEL SLOAN

clear they often did not actually expect the system to be delivered when promised. Indeed, on several occasions I found customers who, from the start of the project, had created plans for their staff expecting a late delivery of the system.

The rules of the game required that everyone pretend the promised date was the real date. When the inevitable slippage became so obvious that it could not be ignored, the pressure and workload rose to intense levels. As I've stated previously in this column, pressure causes teams to work in a very inefficient and error-inducing way due to the high levels of "optional chaos" that too-short schedules generate.[1] Ultimately such projects were invariably delivered late, so while "failure" could not be mentioned or planned for (or even acknowledged after the event sometimes), it certainly occurred with astonishing regularity.

Underestimated Under Pressure

It is interesting to ponder why people and organizations continually act this way. Close up, such behavior resembles the kind of dysfunctional conduct families may display when they are confronted with addictions. A lot of play-acting and denial occurs, and a great deal of energy goes into ignoring, obscuring, and rationalizing the problem rather than observing, acknowledging, and fixing it.

There are other reasons too, which relate to the way we estimate. The most common method I see companies use to create project estimates is to first build a plan for the project. This plan consists of a network of interconnected tasks, with their predicted dependencies and relationships. For each task, the amount of work or effort it will consume and the amount of schedule time it will take is determined. The sum of schedules for the tasks along the critical path gives us the expected schedule. The sum of all tasks gives us the expected total effort, and the number of concurrent tasks at any point in time gives us the expected staffing profile.

For software project estimation, the challenge is in the word "expected." Any estimate, and any plan, is necessarily based on what is currently known. If there are tasks that will need to be executed but are unknown when the plan is created, the plan won't include them. If the tasks take longer, or turn out to have different dependencies than we expect, the plan will not show this. The plan and the estimate created from it can only be based on what we know, not what we don't know.

If we create a plan based on our existing knowledge, it is reasonable to think of that plan as having a "50%" chance of success. That is, if something goes wrong, or there is something we didn't think of that happens to slow the project down or cause more work, we will overrun. If something turns out to be easier than expected, or there are tasks that are not actually needed, we will underrun. These estimates will only hit exactly if either nothing unexpected happens at all or, equally unlikely, the unexpected bad things are exactly canceled by the unexpected good things.

Uneven Chance

If we really did adopt 50% plans, we would expect our success rate in the business of software to be, well, 50%. It is not. It is much lower. It appears to be around 20%.[2]

So how do we get down to 20%? Here are the steps I see organizations taking:

- A plan is created based on what is currently known.
- The summary schedule and effort for the plan is used as the project estimate.
- The planners/estimators are instructed not to include any "contingency," but to give their "best" estimate (whatever that means). However, they do sometimes consciously and sometimes unconsciously insert additional tasks or pad effort to deal with perceived unknowns.
- If management suspects esti-

[1] Armour, P.G. Real work, necessary friction, optional chaos. *Commun. ACM 47*, 6 (June 2004).

[2] A number of sources seem to support this approximate figure. A 2003 study of 421 projects conducted by *Computer Weekly* in the U.K. indicated that only 16% of projects were "successfully completed" (www.computerweeklyms.com/pmsurveyresults/surveyresults.pdf). An analysis of performance studies by the Brussels-based IT-Cortex group concluded that "…about one out of five IT projects is likely to bring full satisfaction…" (www.it-cortex.com/Stat_Failure_Rate.htm).

mates have been padded, they immediately cut the estimate back again. Sometimes, in anticipation of this the estimators pad it even more, which may cause management to cut it even more.

- Over and above this "pad de deux" [sic] dance, management often applies other pressures to force the estimate to be reduced.
- One technique is the appeal to the estimators' and developers' professional pride. In this situation, the manager asserts the productivity will be higher because the developers are much more capable than they themselves are assuming. A friend described this as a management assertion that "…I have more confidence in you than you have in you…".
- Another common ploy is the "assume it's simple" approach, in which the estimators are asked to consider what their answer would be if the system were not as complicated as they really think it's going to be. It is, of course, true that development will not take as long—if the system does turn out to be less complex than we think it's going to be.
- Other assumptions are made regarding the development environment. Perhaps the estimators have factored in a certain amount of project turnover. Then they are asked, "what if the turnover was less, wouldn't the project be done faster?" It would—if there was less turnover.
- The same arguments are made for scope creep, change requests, the effectiveness of development tools, organizational process, and host of other factors.

All of these considerations are true—if the project turns out to be easier than we expect it to be or we are more efficient that we think we will be, of course we will get it done earlier using fewer resources. But the original 50% estimate was predicated on the expected values, not the optimistic values. All these "what-ifs" simply hide the risk; they take the risk from the projected solution and hide it in the rosy assumptions.

The opposite is usually true. Things don't usually turn out to be easier than we think they will—they turn out to be more difficult. What the assumptions do is remove the resources necessary to deal with the inherent risk. But just because we pretend the risk isn't there doesn't mean it goes away.

Rather than adding resources to deal with typical unforeseen circumstances we are actually removing resources necessary to deal with things we *expect* to happen.

To Plan to Fail

So the probability of success for the project starts out at a nominal 50%. Each optimistic assumption that is postulated removes resources from the project and hides the risk in the assumption. This has the effect of reducing the overall probability of completing the project using the assigned resources. Starting at 50%, the probability gets whittled down. When it approaches a 20% likelihood of success, the estimators and team members can finally muster a sufficiently forceful argument that any further reduction is simply too unlikely and that, despite further pressure, they cannot be persuaded to buy into any further reduction. The counter-pressure builds up to a level where equal and opposite force prevents further reduction in the probability.

So the estimate stops at around a 20% probability of success and this is the level at which the average software projects are committed. We know this because of the overall "success rate" of software projects. We know that 80% of software projects fail to meet their goals.

This is an odd behavior for a business because it means we have adopted a typical and consistent business practice that means, if everything works out the way we expect it to, our projects will not fulfill their commitments.

Quite literally, we are planning to fail. C

Phillip G. Armour
(armour@corvusintl.com) is a senior consultant at Corvus International Inc., Deer Park, IL.