

Acquisition Archetypes

Changing Counterproductive Behaviors in Real Acquisitions

Brooks' Law

"Adding manpower to a late software project makes it later."

Brooks' Law is well known in the software engineering community due to the ground-breaking book, "The Mythical Man Month: Essays on Software Engineering" [Brooks 1975].

In this Acquisition Archetype, we look at a program that chose to ignore it—and at the consequences of doing so.

Facing an Aggressive Schedule

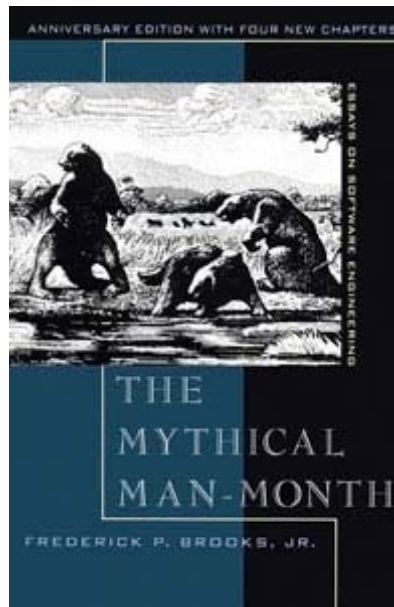
An information technology claims processing program had fallen behind its cost and schedule goals. A new program manager (PM) was scrambling to meet a strict and fast approaching deadline imposed by the program's management review board.

Rose-Colored Glasses?

The PM informed the board that the team could not come close to delivering the latest list of requirements for a November release with its current staff of 50. When board members asked what it would take to meet the deadline, the PM sensed that he had to come up with a solution on the spot, regardless of how realistic it might be. Stressed and hoping that his program could prove to be the exception to Brooks' Law, the PM proposed having the contractor set up a new, additional development site with 20 to 30 staff.

It would cost millions of dollars more.

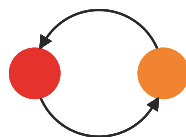
Much later, during an assessment of the program, another manager noted how project stress can force poor decisions. "We bought into the 'mythical man-month'," the manager said, "even though we all knew it couldn't work."



Belief ... and Doubt

Adding a new site certainly was far from ideal. In addition to the added cost, it introduced increased risk. Some managers later called it "the worst situation we could have"—but they, along with the PM, were committed to the aggressive schedule, and adding the site was the way the PM was allowed to add developers.

Expectations for the new site varied greatly. The PM, who had staked personal reputation on the decision to expand the staff, professed it would speed development, moving the project "50-70 percent ahead." Other team members were less optimistic, believing that in the best case they would be no better off—and might, instead, end up farther behind schedule.



More Work, Not Less

The new site was located in Santa Clara, Calif. It was designed to operate for four months.

Once the program started to add developers and ramp up operations in Santa Clara, however, the effect on the program became apparent. It wasn't good. The re-planning was laborious.

"We all knew that it couldn't work."

One team lead understood why the expansion had been done and the pressure the PM faced, but observed that "...ramping up impacted the productivity of the original team." This was true despite the technical expertise of the Santa Clara hires.

The added travel and training duties affected the project leads' efficiency. "The leads ... were only able to operate at 50-75 percent of their normal productivity," noted one manager. Along with the ramp-up came "frustration among the team with the long hours," he said.

Brooks' Law Wins

Flouting Brooks' Law gained the team nothing except budget overruns.

After opening up the new site and growing the staff by 50 percent, one development manager estimated the team got *half* of the November delivery done.

"If we hadn't brought up the Santa Clara team, we probably would have gotten it done in the same amount of time," he said.

(Continued on page 2)

The Bigger Picture

Brooks' Law has been discussed and analyzed extensively in the software engineering literature.

The specific behaviors portrayed by the Brooks' Law archetype include the following:

- geometrically increasing communication overhead that
 - reduces development productivity, and
 - reduces the time available for each individual to do development
- a reduction in experienced personnel available for development (by using them for training of new personnel)

The inner loop of the causal loop diagram shows the effects of peer training, while the outer loop shows the effects of communication overhead. These new tasks give more work to the already overloaded staff. Assigning these tasks adds coordination and replanning time, and more time is lost to “thrashing” as developers switch between training and development.

If the problem that triggers the *Schedule Pressure* (and seems to require additional manpower) is detected late in the program, an over-reaction is likely. This happens because at this late point comparatively drastic steps must be taken for the intervention to have a chance of working before time runs out. Like other management interventions and improvements, there is a time delay before any benefit will be realized by the program. If the benefit occurs *after* development ends, the program only experiences the negative effect, and the effort is not only in vain, but counter-productive.

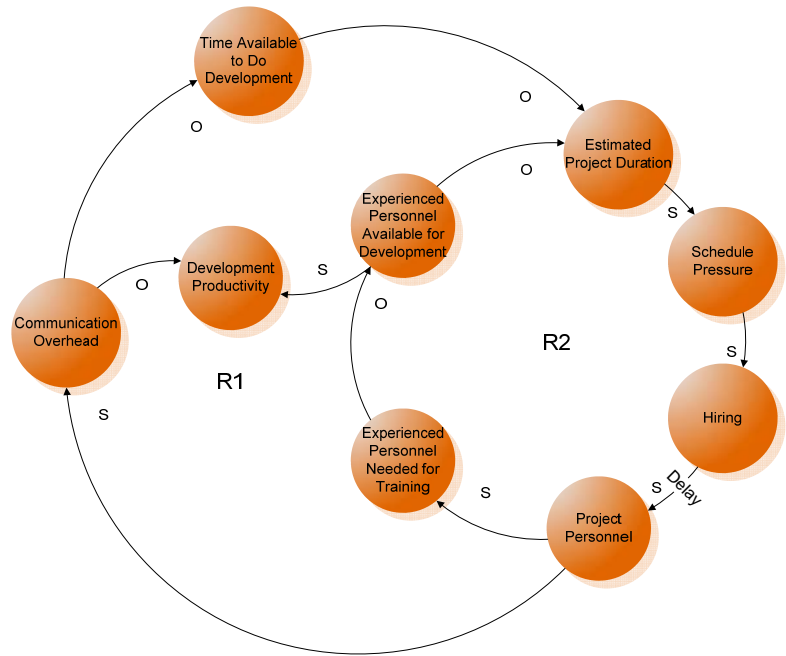
The worst-case outcome is that as *Estimated Project Duration* rises even further, there could be *further* increases in *Project Personnel*, requiring another loop through the diagram. However, this spiral happens only if the organization experiencing it is unable to detect the pattern that it is going on—i.e., if they are unable to learn from their experience on the first iteration.

Breaking The Pattern

Adding staff to a late software project is not inherently a bad idea. The circumstances must be right, however. The key is to explicitly recognize and minimize the unintended consequences of adding manpower as shown in the diagram.

- Adding manpower may be acceptable if there is sufficient schedule in the program to allow it to be done while at the

A Causal Loop Diagram of Brooks' Law.



System variables (nodes) affect one another (shown by arrows): Same means variables move in the same direction; opposite means the variables move in opposite directions. Balancing loops converge on a stable value; Reinforcing loops are always increasing or always decreasing. Delay denotes actual time delays.

same time meeting the intended system scope—so it is important to act as early as possible.

- The scale or degree of the added manpower is significant, as a smaller scale increment in staff will help to minimize the explosive increase in communication overhead.
- The experience of the new staff is also critical; domain knowledge, experience with similar systems and with the development methodology can significantly reduce the need for training.
- Finally, due to the delay before the new staff will become fully productive, the most financially efficient approach to adding manpower is to amortize the investment by keeping the additional staff on the program after they become fully productive. This is not always possible depending on where the program is in its life cycle, and there may be an explicit acknowledgement that it is more important to try to meet a scheduled delivery date than to be cost-effective.

[Brooks 1975] Brooks, Frederick P. Jr. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, 1975.