Acquisition Archetypes

Changing Counterproductive Behaviors in Real Acquisitions

Shooting the Messenger

When a program is in trouble, a responsible manager will want to deliver the bad news to upper management. But people are rarely rewarded for this "whistle blowing," and instead may be ostracized and punished. As is seen in the example below, this has a chilling effect on the other employees and managers, who are then increasingly reluctant to point out any issues, finding it more beneficial to their careers to keep quiet—at least until their tenure with the program has ended.

"Sir, I Have Bad News"

A software development program, underway for several years, repeatedly missed deadlines. So, when the program management declared a firm, "drop dead" delivery date, the team was skeptical. Yet, they kept their reservations to themselves—along with (as the project schedule ticked away)

Executive management was unwilling to hear from subordinates that their assignments could not be completed

the bad news of continuing setbacks and mounting risks to successful completion of the project.

This reluctance and outright fear to deliver bad news grew out of the team's experience with what had become an insular, risk-averse organization—an organization where executive management was unwilling to hear from subordinates that their assignments could not be completed, or that a mandated deadline could not be met. Messengers bearing bad news, one team member said, "were shot."

"Voicing problems or raising risks to management made them [the problems] instantly your fault," he added. "It left you anxious for your position in the organization."

Fear, Uncertainty, and Doubt

Because the upper management wasn't willing to hear bad news, the PM became reluctant to identify or escalate risks until they directly affected releases. This meant the risk focus was usually concentrated on dealing with near-crises, contrary to the premise of risk management—proactive identification and mitigation.

That was punishing to the entire staff.

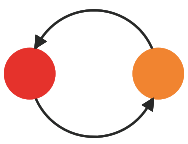
The PM's view was "very much reactive," a team member said. The result was that the program could (1) ignore many risks (i.e., "see no evil") and (2) make decisions that added to the collective risk the program was already facing without having to acknowledge it.

With realistic schedule concerns being ignored, practical alternatives rejected, and having no flexibility to meet the deadline, program management was left with no good options. The PM became resigned to putting on an optimistic face toward the staff, hoping that things would work out by the deadline—despite ample evidence to the contrary.

The PM's view was "very much reactive" … the program could then ignore many risks that wouldn't have to be escalated to executive management

Of course, in this instance, the drop-dead deadline came and went. High-level meetings were held, more dictates were made—and the program stumbled on.

(Continued on page 2)



(Continued from page 1)

The Bigger Picture

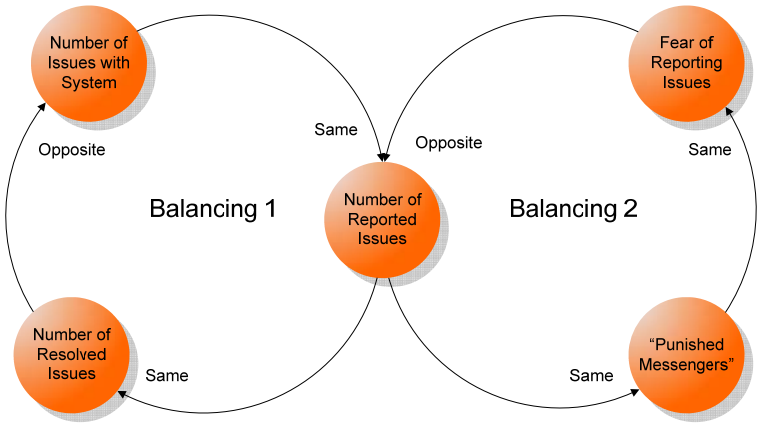
In many programs facing significant issues, it is not unusual to find that team members defer reporting serious risks or problems to upper management, either out of fear of repercussions or in the hope that local solutions might be found. However, there is nothing more damaging for program management than *not* knowing that there are serious issues lurking under the surface until late in the project schedule.

If the program management office staff observe that their program is failing in some respect, they generally feel obligated to inform their manager so that the situation can be addressed. However, that is not the case in some organizations—programs where the messenger may not be rewarded for this behavior, but rather punished. Other staff who observe this result then in turn become increasingly reluctant to point out such issues themselves, finding it more beneficial to their career to keep their issues to themselves. With these perceived negative consequences, the reluctance to deliver bad news increases over time. It's also important to note that this pattern doesn't occur on just one side of the program equation—it can happen on the government side as easily as on the contractor side.

This archetype shows how the behavior of punishing the messenger postpones acknowledgement of more risks into the future, which has direct implications for risk management. Although it is more effective to mitigate risks, and prevent them from becoming problems, programs are more likely to work to resolve actual *problems* (perhaps because they can no longer be avoided). Risks, on the other hand, may be ignored until they *become* problems—but to the detriment of the program.

An important related aspect is that managers may find it more beneficial to keep a program alive until their tenure with it has ended. The comparatively short-term assignments given to program managers in the Department of Defense make simply leaving the program at the end of a tour an acceptable way to "solve" the problems: they become someone *else's* problems.

A Causal Loop Diagram of the Shooting the Messenger effect.



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.

Breaking The Pattern

"Shooting the Messenger" could be attributed to a lack of integrity—or it could be viewed as an indirect consequence of the larger system environment. The archetype diagram shows ways of breaking the overall reinforcing loop at the points where the links can be controlled.

For example, it would be feasible to break the link between *Punished Messengers* and *Fear of Reporting Issues* only if the punishment was insignificant. The only practical point to break out of this dynamic is between *Number of Reported Issues* and *Punished Messengers*—in other words, in the actions of upper management. If this dynamic is entrenched behavior in an organization, it is very difficult to change, because reporting bad news requires trust, and trust is in short supply in such organizations.

A credible signal must be sent to the staff that things are different now, such as the institution of an established risk management process. Once a risk or issue is reported without adverse consequences, trust will begin to build again and the flow of honest information will eventually resume.

To prevent this dynamic, some advocate the independence of status reporting from decision making, "[b]ecause that function is unlikely to provide accurate status if the current status happens to be unfavorable/unflattering to the PMO [Flowers 1996]." Program managers aren't put in the position of feeling they must suppress bad news about the program to protect their careers.

[Flowers 1996] Flowers, Stephen. *Software Failure: Management Failure*. John Wiley, 1996.

Acquisition Archetypes is an exploration of patterns of failure in software acquisition using systems thinking concepts. It is published by the Acquisition Support Program of the Software Engineering Institute.

For more information, visit http://www.sei.cmu.edu/acquisition/research/archetypes.cfm

Software Engineering Institute
Carnegie Mellon

AA12 100114