BPTrends COLUMN

The Agile Practitioner Tom Bellinson February 6, 2018

Stranded on Agile Island

Agile is basically Lean for software development. However, whereas Lean is generally applied to an entire organization, Agile practices often live in an organization surrounded by non-agility. On the surface, that may not seem like a major concern, but for most of the people I have spoken with who find themselves in this situation, it threatens the very underpinnings of the Agile philosophy.

If you are unfamiliar with Agile, it might be helpful to backup and provide some background. Many of the practices associated with Agile are not prescribed even though many organizations apply them. However, there are a few practices that are fundamental. To understand these, let's quickly review how software was once developed before Agile.

Before Agile, the common practice was to spend considerable time and effort to gather all requirements and design the entire system before any coding began. While this approach regularly yielded good outcomes, more often than not, much work was wasted and more work was done than necessary. There are several reasons for this.

- **Environments change** - many complex systems can take months or even years to design. During that time, conditions that drove the design initially may be different or invalid, thus altering the needs of the software.
- **Priorities change** - with a full set of specifications in hand, software developers will usually build the system in the most logical manner, but often the most needed parts of a system are not those that are most logical to build first.
- **Knowing the whole can limit flexibility** - when actual system development begins, knowing what the end product will look like allows decisions to be made that can ultimately restrict the functions of the system. Because Agile developers concede that they don't fully understand the functions of the end product, they are forced to design the system to be more adaptable to new realizations as they are uncovered.

Deadlines

The underlying premise of Agile development is that stakeholders will begin by prioritizing the various functions of the high-level system design. Only the highest priority work will be designed in detail. Further, designs will need to accommodate a

wide range of possibilities in order to support whatever is learned later in the process.

While it could be argued that much of the time projects that are designed in toto before actual software development begins only appear to have fewer unknowns. In fact, the same number of unknowns is there, but without Agile, commitments to the path of the known cause greater effort to undo them. However, this myth of knowing what is ahead provides comfort to those around the project.

This brings us to the first point about being "Stranded on Agile Island." Good Agile practitioners know that in order to address uncertainty, teams cannot fix both time and scope. Either, the functions of a particular piece of software are fixed and the time allotted is allowed to float. Or, the time allowed is fixed and the scope of work is allowed to float. When both are fixed, there are two underlying expectations:

1. Every aspect of the software being written is completely understood
2. The capabilities of each team member is completely understood
3.

Even a cursory understanding of software development would inform that it is unlikely that either of these conditions can be met in all but the rarest of situations. Nonetheless, non-agile stakeholders will often press for a commitment to complete specific functionality on a specified date. These stakeholders will often see "we're Agile" as an excuse for incompetence or at least an inability to plan properly.

If the disconnect between external stakeholders, who are usually organizational leaders, is not addressed, Agile teams may be forced to do the type of extensive planning that has been shunned by Agile practices. Thus, the efficiencies that can be gained are undermined.

The Mythical Man Month

Back in 1975 Frederick Phillips Brooks wrote a book entitled The Mythical Man Month: Essays on Software Engineering. The book recounts the experiences of the IBM team that was developing a new mainframe operating system for the 360 series computer. For our purposes the important point here is that with a commitment to both a defined set of functionality and a deadline, the project spun out of control and caused the timeline to become far more extended than necessary.

The reason for this is that as the deadline approached and it became clear that the work would not be finished, leaders of the organization felt the need to take action. The obvious choice is to add people to the project to speed things up. Unfortunately, as anyone who has experienced adding someone to a project that is well underway knows, the initial effect of doing this is to slow the project down. Why? Because new people need to come up to speed. The more complex a project is, the more effort will be required for them to become fully productive members of the team. That "effort" is expended by fully productive members of the team who must now spend their time training, rather than moving the project forward.

Eventually, work velocity increases, but depending on how much of the project is left, it could be hard to recover the lost productivity. Furthermore, as people are added to a project, more coordination is required. This is sometimes referred to as communication overhead. This overhead must be deducted from team productivity, which means that the per person output goes down.

What Brooks showed is that had the project proceeded with the original team, it would have most likely been completed sooner than when people were continually added to staunch the ever-decreasing individual productivity of a team comprised of too many short-timers.

Trust

Deadlines are easily the biggest danger of trying to run an Agile team inside a non-agile organization. However, it is by no means the only one. Another common issue centers around trust. Good Agile teams work hard to create an environment in which participants learn to trust each other. By continually evaluating performance issues and experimenting with ways to address them, good Agile teams build confidence over time which translates into trust that both the people and the processes employed are going to be effective and efficient.

This process of continually retrospecting on what is working well and what is not can (and usually does) cause this part of the organization to function better than other parts. People who operate outside of an Agile environment rarely build the level of trust within one. The effect of this is that practices and processes are put in place to compensate for a lack of trust. These practices and processes can infect the trust built by Agile teams.

Even though teammates may trust each other, they are forced to put CYA type practices in place to shield themselves from other parts of the organization. These practices do not contribute to finishing the work at hand and can even derail the benefits of having a trusting group by "building in" distrust.

Things to Try

If an Agile team finds themselves in this situation, it is important to see it just like any other impediment to team performance. Meaning, a plan should be put in place to address it and that plan should be iterated until success is found. Maybe educating leaders in the organization can be effective. If they are unreceptive, another strategy is to convince a non-agile group within the organization to give Agile a try. If they can demonstrate productivity improvements as a result, leadership may then take notice.

Every organization is different. In my own, it so happens that, as I write this, next week we are hosting key process stakeholders from non-agile parts of our organization to demonstrate some of our Agile practices. In particular, something called a blameless postmortem (or BPM), in which teams review an incident or failure with the primary purpose of learning of systemic issues that can be avoided in the future. By intentionally avoiding blame, we can encourage people involved to share their experience freely without concern of repercussions. Only through this frank analysis can maximum benefit from the experience be gained.

The other process that we will be sharing is the retrospective. Retros, as we like to call them, are simply a regular space for teams to consider the things that helped them be successful so that they can continue doing them or even expand the behavior beyond current practices, and to consider things that did not go so well and what can be done to avoid these negative situations in the future.

Even though our organization is already fully supportive of the Agile teams and respects their practices, by helping other teams become more Agile, we believe that we can not only improve overall performance, but we can increase trust throughout the organization (which happens to be one of our core values), which should bring additional efficiencies to everyone.

Tom Bellinson

Mr. Bellinson has been working in information technology positions for 30 years. His diverse background has allowed him to gain intimate working knowledge in technical, marketing, sales and executive roles.

Most recently, Mr. Bellinson finds himself serving as President of a BPM related software start-up company called UnaPage that provides solutions based on Microsoft SharePoint. From 2008 to 2011 Bellinson worked with at risk businesses in Michigan through a State funded program which was administered by the University of Michigan.

Prior to working for the University of Michigan, Mr. Bellinson served as Vice President of an ERP software company, an independent business and IT consultant, as chief information officer of an automotive engineering services company and as founder and President of a systems integration firm that was a pioneer in Internet services marketplace.

Bellinson holds a degree in Communications with a Minor in Management from Oakland University in Rochester, MI and has a variety of technical certifications including APICS CPIM and CSCP.