BPTrends COLUMN

The Agile Practitioner: Business Process in an Agile Age Tom Bellinson February 2020

The Agile Manifesto states: "Individuals and interactions over processes and tools." Here at BPTrends, we care a lot about processes and tools, so this may be a bit disheartening. In fact, I have heard more than a few agile practitioners call this out when they don't like what they see. The call is usually sounded when processes become too rigid.

The key word in the statement quoted above is "over." As someone who deeply understands the value of a good process, I find myself reminding people that it doesn't say "instead of." The progenitors of agile practice understood well that tools and processes are important. Many of them came from organizations that had gotten so enamored with processes and tools that they had all but completely forgotten about individuals and their interactions. They must have known that there was no way to eliminate processes altogether, nor would they have wanted to.

The statement reminds us about the importance of collaborating. True innovation often requires the breaking of current processes and throwing out current tools. Innovation is at the core of agile practices, so this stands to reason.

Process Innovation

Ironically, the agile evolution has caused a groundswell of new approaches to process management. Some processes are expected to evolve so quickly that the type of rigid automation we were moving towards 15-20 years ago would no longer be tolerated. Yet, some processes and tools seem to be firmly entrenched.

For example, consider the "scrum" brand of agile practice that most organizations employ. Teams execute in what is commonly called "an iteration" or sprint. The iteration was once as large as one or even two months. At that point in time, most companies had been operating on six month cycles or even longer, so this seemed quite aggressive. As teams moved along on their agile journeys and agile coaches gained more experience, the time frame for these iterations got shorter.

However, what hasn't changed is the basic process of an iteration:

1. Design and detail the work (into user stories)
2. Plan the sprint (or iteration)
3. Hold daily stand-ups (so individuals can interact)
4. Do the work (more on this shortly)
5. Hold a retrospective (to reflect on the good and the not so good)

...and the cycle repeats. It usually repeats for years with little modification to the basic process. Despite the call for continuous improvement, this process seems to be here to stay.

Other parts of how work gets done change almost continuously. One might suspect that a process, the way we traditionally think of it, would be useless in the face of such rapid change, on the contrary, this environment has forced the lowly process to evolve faster and in better ways than it may have without this pressure.

It has been a while since I was involved in the BPMS space, so for the writing of this Column, I visited the website of one of my favorite tools in the BPMS marketplace, Appian. In large letters on their homepage, they boast of a client delivering their first application in 6 weeks. For those of us who have been around for a while, this seems really fast.

However, software development teams are now developing applications in a few weeks or even days. At ITHAKA, we deploy multiple new features to our system daily. Many of them are very small, but they deliver incremental value to users sometimes minutes after they have been completed. We are constantly pushing ourselves to go faster. This means that processes also need to evolve quickly to keep pace with the evolution of the technology and practices which drive our software development.

People Processes

While most organizations have relatively static processes, those practicing agile management techniques take a much more decentralized view of process management. Agile teams work best when they are mostly autonomous. The fewer dictates from above, the better. This leaves each team to find their own best practices and continue to experiment to find better ways of doing things.

Due to the nature of the sprint iterations that most teams use, processes are evaluated more regularly than in a traditional environment. In the sprint retrospective, everything is up for grabs. Whenever someone calls out an opportunity to make adjustments to an existing process, the team is free to examine alternatives and create experiments to try now approaches.

We call these changes "experiments" because we want to remind the team that any change we make is reversible or modifiable if we don't like the results. Because we are constantly tweaking and adapting our processes, we do not have formal documentation for many human practices which the team repeats. We do keep a list of what we call "team agreements." This list does not represent all the processes that have evolved, but it allows us to onboard new team members more efficiently and remind ourselves of how we have chosen to operate.

Automated Processes

The other class of processes which is in constant flux is our automated processes. As mentioned earlier, we deploy new features daily. To make this possible, we have a highly automated pipeline that allows a team to go from coding, to testing, to installation in the production environment and final verification. We continually strive to make this process more robust and hands-off. When new features are added, new tests must also be created to ensure those features will continue to work properly

when subsequent changes are made. These new tests become part of the automated process of delivering new features.

Thus, our deployment pipeline process is continuously being modified to ensure that we can rapidly deliver new features. And, it's not just the deployment process that is regularly being modified. Developers are always on the lookout for new tools that can help us deliver better code faster.

Recently, we started working with a new tool called GraphQL. I will try not to get too technical here, but some details will be required to make the point. Our system is made up of hundreds of small applications called microservices. These small applications typically have one or a few functions each and can therefore be reused by multiple other applications. So, for example, if one of our many user interface applications wants to authenticate a user into our system, they would call the one application that handles this function.

The traditional challenge with this is that each user interface application that wants to authenticate a user may have slightly different requirements. In order to accommodate this, the authenticating application needs to evolve to provide the necessary information for each different user interface application. Over time, the payload of information coming from the authenticating application gets bigger and bigger as these new requirements are added. Now, every user interface application that needs to authenticate has to receive the large payload of data that is more than it needs and could potentially cause issues.

GraphQL has given us a way to manage the payload of data such that each user interface application can get just what it needs. This makes the system faster and more stable because the authentication application can evolve as needed without impacting user interface applications that haven't changed. Those will still get the exact payload they were expecting.

Keeping it Lightweight

The key for both types of processes is that we do not have a lot of overhead associated with change. Teams are free to implement new approaches or tweak existing ones in order to deliver faster, better and less expensively. It is in our DNA to be constantly pushing on all of these levers of progress. We cannot allow ourselves to get bogged down in hefty vetting to ensure these process changes will go the way we hope.

We try things and if they work, we keep them and if they don't, we pivot with little fanfare. This practice does not lend itself to extensive process flowcharts or even detailed documentation. This is not to say that we don't have extensive technical documentation about the architecture of our systems and the "contracts" between different applications of how they will communicate with each other. We do. These documents are very dynamic and change regularly. Often, a few conversations will happen between the teams that are impacted by a change, the documentation is updated, and the changes are rapidly implemented. It's still about individuals and interactions over processes and tools, but all of it is important.

Author

Tom Bellinson

Mr. Bellinson has been working in information technology positions for over 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 a Scrum Master for ITHAKA, a global online research service. 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.