Software Development: An Agile Journey

In the last installment of the software development series, I provided some history and perspective on how software gets developed. This brings us to the state-of-the-art in the process of software design and development.

Most industry insiders refer to the various modern approaches to software development as Agile. The two most common variants of Agile are Extreme Programming (XP) and Scrum. XP actually predates the term Agile. Kent Beck is commonly credited with inventing XP. His book, Extreme Programming Explained: Embrace Change, was published in 1999.

The differences between the various Agile methodologies are minimal. They all conform to the tenets set forth in the Agile Manifesto (to which Kent Beck is the original signatory). Let's review the 12 principles of the Agile Manifesto through the eyes of a business process practitioner. We will also take note of the connection between Agile and Lean thinking.

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Here we see that identifying the customer and their needs is paramount in establishing value. This is a fundamental expectation of any process improvement effort, so there is no surprise that the progenitors of the Agile Manifesto would put this as their #1 principle. In the Scrum methodology, there is a role called the "product owner" which defines a single person who is responsible for determining "value." This role is very similar to the "process owner" in process management initiatives.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Well, it would be rather awkward not to incorporate a statement of the value of agility in an Agile Manifesto, so this is a good choice for the #2 principle. This principle speaks clearly to the need for continuous improvement and, like Lean, places a premium not only on learning, but "applied" learning. Again, the emphasis is on viewing the benefits of change through the eyes of the customer. Don't worry, the benefits of the team are considered later.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Here we see another critical tenet of Lean in play. Small batches are a critical component of agility. In The Lean Startup, Eric Ries talks about how each delivery cycle provides an opportunity for learning from the customer. By shortening the interval between each PDCA (or PDSA as Deming originally proposed) cycle, teams can reduce waste by spending less time building the wrong things. It's better to throw out a week's effort than six months' effort.

Business people and developers must work together daily throughout the project.

Anyone involved with process improvement initiatives knows that the people involved with the daily activities associated with a process are the best source of knowledge. Almost all processes have what is often referred to as the "happy path." This is what happens when all goes well and as expected. With any luck, most processes will follow the happy path most of the time.

The rest of the time, alternate paths will be traveled by process stakeholders. If they have been executing the processes for a significant time, they will be familiar with how to navigate these exception paths. Most of the time, these deviations have been addressed before and there are procedures for their redress. Will these procedures be well documented? Maybe, but probably not.

The best way to understand the full range of variations a process can take is to work closely with the people who live them. When developers work side-by-side with process stakeholders, process variations can be identified and incorporated in small bites as the product evolves. If developers are forced to work apart from the process experts, they will make bad assumptions and spend many hours realizing those bad assumptions in their software. The best way to avoid this is to close the gap between making a bad assumption and correcting it.

Build projects around motivated individuals.

If Agile has one big divergence from common process thinking it is that the philosophy encourages focus on people and communication over process and tools. Naturally, process managers tend to spend a lot of time thinking about their processes and the tools that support them. However, they are not oblivious to the importance of the human factor.

In fact, it is quite possible that this difference is only one of semantics. What underlies this tenet is that processes must be malleable in order to adapt to changing environments. More specifically, as project teams are created to develop software, the members of the team will have a unique combination of skills, abilities and motivations. How they work together should be decided by them. This self-organizing principle is at the heart of the Agile philosophy. Organizations that impose rigid processes on project teams risk limiting their full potential. Giving teams the ability to self-organize creates motivation to succeed.

Give them the environment and support they need, and trust them to get the job done.

There are many books on leadership. One of my favorites is Good to Great by Jim Collins. In the chapter on leadership it is suggested that good leaders provide clear communication of the mission and vision, put the right people in the right seats on the bus, and get out of the way. In this principle, Agile is simply advocating for good leadership.

It is especially important for Agile just as it is in Lean. Here again we see a parallel that suggests leaders should push decisions down as far into the organization as possible. This also suggests that training and "servant leadership" are essential components of effective teams.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

As I mentioned in my last Column, the traditional Waterfall approach to software development relied heavily on rigid and detailed design documentation as the means for knowledge transfer. History has already shown that this approach is ineffective. As a species, we learn to listen and talk long before we learn to write or draw pictures. Language skills are built into our DNA. Some people communicate effectively using the written work – others do not. This principle suggests that it is unwise to put a project at risk simply because some members may lack communication skills in forms other than spoken language.

Furthermore, much secondary information is gained about the importance of ideas from the non-verbal aspects of face-to-face communication. At the University of Michigan we have one team member who works and lives in Ohio. We have a telepresence bot that he uses to be a part of the team. It is essentially an iPad on the end of a remotely controlled Segway (it's not really a Segway). It allows the team member to "be in the room" when we have team discussions. He is able to look at people and they can look at him. One time, I caught him (rather his telepresence) watching a game of foosball through the window to the game room. Some things can't be done remotely. Fortunately, he gets a chance to actually play when he makes his periodic visits to Michigan.

Working software is the primary measure of progress.

Here, we start talking about metrics. While there is no doubt that even Agile development teams must spend some time gathering requirements and designing their product, these activities ultimately delay the feedback loop that comes from building software and having end users try it. Only at the intersection of real software and real users can true learning be achieved. Therefore, using working software as the primary metric for progress keeps everyone focused on the proper goal. This is not to say that there can't be other metrics and other learning opportunities, but this one represents the core purpose for software development teams.

Agile processes promote sustainable development.

This sustainability principle is likely borne out of a long history of software development teams falling behind and then being asked to work overtime until the project is caught up. As most who have had this experience will attest, the normal result is physical and emotional stress that reduces individual productivity. The lack of proportional productivity gains creates additional stress, and projects spiral out of control as individuals start leaving under the duress of a project doomed to failure. One of the benefits of engaging the business stakeholders with the development team is creating a shared sense of ownership over progress. It encourages everyone to manage the project cadence so that undo stress does not undermine said progress.

The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

For developers, creating software IS their work, but for sponsors and users it is a diversion from their normal work. As mentioned above, the pace of effort must be sustainable. This principle reinforces the importance of sustainability by extending it to ALL project stakeholders and not just the development team. Here, we can invoke the Theory of Constraints to note that resource bottlenecks will have a major influence on who is setting the pace of the project. It may not be within the development team.

Continuous attention to technical excellence and good design enhances agility.

This is the most elusive principle of all. How many times has a leader said, "we are the best…?" Fill in the blank. What does it mean to be the best? Hiring the best people? Offering the best training? Continually creating opportunities for personal growth? Using the best tools? Chances are, it's all of the above. The challenge is that technical excellence is an investment that takes people away from the primary metric – making working software. These investments are costly and painful in the short term. In the long term, they are essential. This principle is embodied in Lean and Six Sigma which both strive to continually seek improvements.

Simplicity--the art of maximizing the amount of work not done--is essential.

This principle may be somewhat unique to software development. Few products are as malleable as software. In The Lean Startup, Ries introduces the idea of the minimum viable product (MVP). The idea is to figure out how to create something useful as quickly as possible. This runs counter to the instincts of most people, who are inclined to make things that are complete.

Like Lean, small batch sizes are recommended. By creating an MVP and then adding functionality in the smallest possible chunks, the feedback loop between developers and users is shortened. While it is tempting to add desired functionality, you might just learn that it is not necessary when users never ask for it once the product is in play.

The best architectures, requirements, and designs emerge from self-organizing teams.

When we talked about motivated individuals, we invoked the idea of self-organizing teams. These two principles are two sides of the same coin. If motivated individuals do the best work, then self-organization is one of the best ways to create motivated individuals. Agile practices encourage team members to be called…well…team members. When we distinguish between programmers, business analysts and project managers, we miss opportunities to get the most from people with diverse skills. Many programmers are good analysts and many analysts and managers are good programmers. When teams exploit all available talents and skills, everyone is more fulfilled and few important ideas are overlooked.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The Scrum methodology recommends a regular meeting called a “retrospective.” The idea is create a space for self-reflection. In this meeting, three questions are asked of the team:

1. What did we do right (good)?
2. What did not go right (bad)?
3. What can we try to do better (try)?

This encourages the team to acknowledge progress, but not ignore issues that are inhibiting greater effectiveness as a team. In our retrospectives, we limit #3 to things that we can at least start to try in our next iteration. Grand schemes often take too long to implement and often become stalled before completion. By eating our elephants one bite at a time, we find ways to continually improve that make a difference week after week.

Summary

Much has been written about Agile methodologies, just as much has been written about Lean. These writings can tell us where to start and guide us if things seem to be going off track, but they cannot provide the kind of roadmap that we find in other disciplines. Like Lean, Agile is a journey. In particular, it is a personal journey that teams must take alone. The organization can give permission for them to take the journey, but then it must get out of the way and allow the team to follow their own path.

This is a difficult thing for leaders to do. Leaders often feel obligated to set direction and make decisions. The challenge for them is to determine which directions and decisions they are uniquely suited for…and stay out of the way of the rest. Employees rarely make it easy. They come to leaders seeking advice. Sometimes, the best advice is: “go figure it out for yourselves.”

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.