Essential Ideas About Agile Methods

By Scott Duncan

www.westfallteam.com

My involvement with standards, methodology and process issues brings me in contact with people who ask what agile methods are or what they are about. Many have not heard of the Agile Manifesto and associated principles. When I ask what they have heard about agile methods, I am told things like pair programming, no documentation, refactoring, iterative development, no planning, daily “stand-up” meetings, etc. Of course, none of these get at the heart of what agile methods are “about” (beyond the fact that some are not true).

Based on listening to the creators of agile methods and reading what they have written on the subject, agile methods, above all else, seem to me to be about expanding the bandwidth and frequency of communication and about being open to change. Essential to both of these is the existence of effective feedback, which is required for meaningful communication and productive change. The values, principles and practices of agile methods seem to flow from seeking to achieve these objectives.

Nothing improves communication (especially about change) more than regular feedback through direct contact with the customer, the development team and the software being created:

The Customer

To improve communication with the customer, have the customer on the development team or as close to it as possible. The customer can then participate directly in specifying needs, defining acceptance criteria, regularly reviewing functioning software, and providing rapid feedback on what is and is not needed while learning more about how to achieve those needs. As the distance between customer and developer grows, both physically and through less frequent feedback, the bandwidth of communication shrinks. It takes an increasingly greater commitment of effort to overcome that distance. At some point, the distance can increase to where people feel there is nothing that can be done to overcome it. In the face of that, passing very formal documentation back and forth replaces more effective communication. Indeed, when this happens, true collaborative communication is replaced with various forms of negotiation formality.

The Team

Similarly, to improve communication within the development team, all members should be physically collocated so regular face-to-face communication is possible. Then brief, daily meetings of the entire team can be held, so everyone knows what everyone else has already accomplished and, likewise, will accomplish that day. From a planning perspective, an iterative development lifecycle and commitment to meeting each near-term iteration plan should result in the least delay in or confusion over what is to be delivered to the customer. The team can then deliver a functioning release frequently -- every 1-4 weeks is common -- and the customer can help run and validate the results of acceptance tests offering feedback on what does and does not meet business needs.

The Software

Finally, to improve communication with the product itself, so we know if we are achieving our plan and meeting customer expectations, seek frequent feedback from the product. Do this by building and testing the system at least daily to know immediately what problems exist and what progress is being made. Then, deliver a functioning release frequently, getting feedback from the customer to learn more about how to achieve their (evolving) needs. To do this requires an environment, technology and team commitment to support iterative development and rapid turnaround (with effective configuration and quality control) to achieve the frequency required to adapt to change easily.

From the desire to improve communication and respond to change, flows the values of the Agile Manifesto (www.agilealliance.org) which guide the agile approach to software development:

1. Individuals and interactions are valued over processes and tools
2. Working software is valued over comprehensive documentation
3. Customer collaboration is valued over contract negotiation
4. Responding to change is valued over following a plan

Those who developed these values have said, “While we value the items on the right, we value the items on the left more.” Indeed, the items on the right may be viewed as valuable only in so far as they effectively support the values on the left. If the items on the right do not do so, they may even become harmful to achieving the items on the left. So let’s look at the right side of the value statements in light of communication and change:

Processes and Tools

The point in having processes and tools is to help ensure that otherwise skilled people are supported in their efforts to do a good job by removing from them the responsibility for reinventing, remembering and repeating many routine, but necessary, tasks. If you view the role of processes and tools as one of trying to prevent people from doing harm, you’ll probably want as much process, as little change, and as few independent choices as possible. Bluntly, you don’t fully trust people and believe process needs to exist to “make sure” (some would say “force”) people do the “right” things.

Documentation

Similarly, the point in having documentation is to communicate, to someone, what is happening (or has happened) on a project. Unfortunately, comprehensive written documentation provides the lowest bandwidth of communication while placing high maintenance demands on the development team and customers. Developers usually do not like to write documentation and may not be good at it even when they try. It is also hard for others to keep up with projects from a distance. So the goal has to be how to document what is needed as a part of the process of producing working software, not as a parallel activity. A lot of documentation is viewed as necessary (a) to confirm what is happening through milestone reviews between parties who are not communicating regularly in other, better ways or (b) to make sure someone, at a later time, can figure out what happened during the project and what the system “does.” Various agile practices attempt to address this in ways that add to understanding of the project while reducing maintenance overhead.

Contracts and Negotiations

Agile methods seem to view contract negotiation as necessary and perhaps even useful, but definitely insufficient to ensure an effective working relationship between developers and customers. Formal agreements offer “boundary conditions” and, for some, a sense of “insurance” if things don’t go well, but they are, after all, just another form of written documentation with all its associated problems. Unfortunately, dynamic things like requirements and schedule are often bound up as part of such contractual situations making it at least hard, if not nearly impossible, to officially respond to change productively. However, you can rarely, “buy software the way [you] buy a car,” i.e., by combining a set of options already built into the structure and pricing of the product. Most non-COTS

> (Commercial Off-the-Shelf) software is custom-built and, often, demanding that we have to define everything completely up front suggests people have but one chance to "get it right." Almost by definition, such an approach increases the communication distance between customer and developer, reducing the bandwidth to large, formal, written documentation.
>
> Plans
>
> Plans have to recognize the need for, respond to and accept change. Few successful projects tend to deliver just what was described in the original plan and only that. Belief in the validity of very detailed long-term planning may just be a way to kid people into thinking they are ensuring success. It's another form of formal agreement and documentation that, in and of itself may not be bad, but cannot be relied upon to reflect actual needs over time. On the other hand, having a succinct, shorter-term plan that is not interrupted nor deviated from can produce expected results rather effectively: developers and customers can likely agree on what can and should be done in, say, a couple weeks to a couple month's time. Beyond that, planning just seems to have to be less precise. Indeed, a long-term plan gives people more of a chance to "hide" behind lengthy milestone periods and encourage doing things in a rush rather than with constructive urgency. It can allow for the "90% done" syndrome where "all-nighters" are used to meet a deadline because shorter-term progress (or lack of it) hides behind less regular communication and evidence of achievement and progress. In longer-term planning with "big bang," end-of-schedule deliveries, success gets measured along the way in terms of documentation produced rather than functioning, high-quality software.

The values of the Agile Manifesto, and the principles associated with those values, may not be easy (or even possible for many reason) for everyone to implement. However, if you think your organization would benefit from expanding communication bandwidth and frequency as well as being more able to adapt to change, consider what agile methods have to say about the value of encouraging and supporting through direct contact and regular feedback.