Being Agile Without Going Overboard

Venkat Subramaniam
venkats@agiledeveloper.com

Abstract

Agile Software Development is gaining popularity. But, what does it mean to be agile? Is it using unit testing, continuous integration, following XP, Scrum? In this article we discuss how to introduce agility into a project in trouble and not currently agile.

Agile Methodologies

Quite a few agile methodologies have surfaced over the years–Extreme Programming (XP), Scrum, Crystal, Lean Development (LD), Adaptive Software Development (ASD), Dynamic System Development Method (DSDM), Feature Driven Development (FDD) to mention a few. While these methodologies have variation in their emphasis, some of the common themes among them are: succeeding in development, letting the design evolve, creating robust code, and above all, seeking feedback by interacting with customers.

Different methodologies have come to mean different things to different people. Some think that you're not doing XP[1, 2] if you're not following–by the book–all the core practices of XP. XP has important practices. Scrum has important practices. In addition, there are other important practices to pickup and succeed[4] as well.

Developers ask "Which methodology should I follow?" They ask "Should I use XP, should I use Scrum, what can I do to turn my project around?" Very relevant questions and we'll address these in this article.

Introducing Agility in a Failing Project

Methodologies often come with ceremony. Some heavy weight processes can be very ceremonial by requiring you to follow a set of steps, developing certain set of documents, etc. Certain amount of ceremony is good, provided it truly influences your success. Agile methodologies generally expect that you practice unit testing, hold standup meetings, etc. These are ceremonies as well and agile proponents (including your humble author) will say that these are good ceremonies to follow. But, are these the first set of practices you should take up? Should you take all the practices presented to you? Should you take them all at once? No, certainly not.

Let's draw an analogy. We can agree that eating right and exercising are good habits to maintain good health. However, if a patient is brought in complaining about strong chest pain, you would not (want to) hear a doctor say "If you had eaten right and exercised well, you wouldn't be here. So, get up and run now on this treadmill." That would be absurd and may prove fatal. The patient has to be first stabilized, his condition improved, before being put on a regimen.

If you are part of a project that is in trouble, full-fledged agility may prove to be fatal to your project as well. Let's look at some practices that can help restore the project and place it on the patch of recovery, before introducing other nice practices.

Iterative steps towards agility

Assume only a few months are left to finish your project, but your team is way behind. Furthermore, assume that your team is not familiar with most of the significant agile practices like unit testing, continuous integration, etc. Introducing unit testing to such a team may require time to understand it and to get better at it. While unit testing has significant benefits, now may not be the right time to introduce to your team.

A gradual (iterative and incremental) step towards making your team agile may prove to be prudent. The first step before solving a problem is to understand what the problem is. Why is the team behind schedule? What's holding them back? All the good practices in the world can't help your project–all at once–at least. It is important to understand what the major and imminent problem is and address that first. Once you have addressed the major problem(s), then you can move on to making improvements and adjustments.

An Experiment with Iterative Agility

I had an opportunity to be part of a project in crisis–a prospective client asked help revive a project in trouble. The project had gone too long with little success. The team didn't follow any agile practices. They weren't communicating. There was no iteration cycle. They were struggling to keep up with the schedule, trying to balance between writing new code and fixing bugs.

They only had a short few months to finish the project; the team was in panic, the manager in despair. Asking the team to take up, for example, unit testing did not seam like a prudent option at that time. What are the minimum agile practices that I can use to turn this project around? We decided to start with three practices that seemed the most logical at that time, given the state of the project and the team: Weekly iteration and Demo, Daily standup meeting, and Prioritization and Tracking.

Weekly iteration and Demo

The team was nose down, frantically trying to get things done. Any time they looked up the scope of the project and the list of tasks were overwhelming. There were some legitimate concerns among team members. Should I be fixing the bugs? Add new features? Which features? What about the ones I started working on last week? Two weeks ago? You want me to handle all of it, now?!

It would certainly help the team if they can clearly focus on what they are support to work on. By deciding to work on week long iteration (different methodologies recommend one to six weeks of iteration duration), we were able to define the scope of work for one week at a time. It gave each member of the team and the manager an opportunity to sit down and decide the objective for the week. Once the list of task for the iteration was decided, everyone on the team reviewed it. We did not want to setup for failure–the team had to agree we're looking at what's reasonable to achieve.

At the end of the week we held a demo to key customers who would provide critical input on the project. It was the goal of the team to meet the expectation of the customers at this demo, one week at a time. During the demo the team showed what was actually built, illustrated the features they have completed, and sought input on what can be

improved or need to be changed. We also decided what should be the priority of tasks for the next week.

Daily Standup meetings

The team members were saying that they were simply taking marching orders and did not know where everyone was. People were not communicating with each other.

The second practice we introduced in this project was daily standup meeting (introduced in Scrum[3] and adopted in XP). Each team member had the opportunity to speak for a short duration. There were required to keep the focus on saying three things: What they did the day before, what they plan to work on that day, and what–if any–is holding them back.

This gave an opportunity for the team to know what everyone was working on. It also helped prepare for the day. It helped bring focus to what each developer had on hand for the day.

Prioritization and Tracking

The team was having trouble keeping focus on what they should be working on. The development effort was like “catch the crab” game–they ran to fix problems that appeared to pop in random. They were using email primarily to report problems. Unfortunately, these emails were often lost or buried under the heap of other emails and spam. By using a simple tracking system, we were able to enter tasks and problems, assign priorities to those, and check on the progress made by developers. It brought some sanity.

Other Practices

We went on to introducing some other useful practices on this project, but definitely not all at once. The goal was not to be agile, but to succeed. We did things in a way as to improve the confidence of the team and show that we were making progress. We rejected or postponed practices that did not promise immediate benefits. We saved practices with long term benefit for a later time. By following a select set of practices that made sense and with the support and genuine interest of the team to succeed, the project was completed ahead of schedule–something that was unimaginable at one point on this project.

Conclusion

Ron Jeffrey’s put it wisely and aptly: "There is no award for 'being XP.' There is an award for doing the right combination of practices: success." Focus on succeeding rather than trying to be agile by-the-book (any book!). First, pick the practices that actually solve your problems and show progress. Then move on to other practices that will make you and your team better. How do you know you are doing it right? There is no better proof than success.

References

1. Kent Beck and Cynthia Andres, “Extreme Programming Explained: Embrace Change," Addison-Wesley Professional.
2. Ron Jeffries, Ann Anderson, and Chet Hendrickson, "Extreme Programming Installed," Addison-Wesley Professional.
3. Ken Schwaber and Mike Beedle, "Agile Software Development with SCRUM," Prentice Hall.
4. Venkat Subramaniam and Andy Hunt, “Practices of an Agile Developer,” The Pragmatic Programmers.