Uncomfortable With Agile

Andy Hunt, The Pragmatic Programmers, LLC

Abstract. The original intent behind the agile software development movement and its goals have become diluted over the past decade, with many teams simply doing selected agile practices instead of becoming agile practitioners. Many adopters and their organizations now claim to be comfortable with the idea of agile software development and their own use of it, which is probably an indication that they are doing it wrong.

Introduction

The Agile Manifesto was created more than 10 years ago, in February 2001, by a group of 17 interested individuals. I was fortunate to be one of those 17; witnessing and participating in the articulation and advancement of an idea that would, we hoped, change the face of software development for the better. But 10 years is a long time in the real world, and it is an eternity in the tech world.

As a result, things did not quite turn out the way we expected. In 2001, when we created the manifesto and launched the agile software development movement, we fully expected to see a crowded rush of new methods, new practices, and new ways of being agile. These new ideas would follow the guidelines of the manifesto but introduce new ideas and approaches and help advance the art. That did not happen.

Instead, we have seen wider-scale adoption of individual practices that agile software development favors, including formerly controversial practices such as pair-programming, and even basic hygienic practices such as version control that previously were not always followed. Overall, this seems to have had a very positive effect on many development organizations, but it is surely less than many of us had hoped for.

It seems that the initial focus of the agile software development has been forgotten over time, and the goal has shifted from becoming adaptive, flexible, agile practitioners to merely following a subset of canonical, prescribed agile practices. It appears that people have forgotten why we do what we do.

Of course it is not just software development that suffers from this sort of phenomena. For instance, when Osama bin Laden was caught and killed in May of 2011, there was a huge spike in search traffic asking, “who is Osama bin Laden?” Some two-thirds of the searchers were teenagers, which casts serious doubt on our ability to transfer information to the next generation [1]. We, as a people, had forgotten why we were doing what we were doing.

If our collective mass memory can forget such significant, pivotal events in our shared global history, then remembering the subtleties of a niche topic like “effective ways to develop software” is probably doomed from the outset.

So it is not too surprising to consider that perhaps the original intent behind the agile software development movement and its goals have become diluted over time, and perhaps have not been transferred to our peers, newcomers to our teams, or our successors. It is not surprising that many adopters and their organizations now claim to be “comfortable” with the idea of agile software development and their own use of it.

If you consider yourself comfortable with “agile,” then that is too bad, because it probably means you are doing it wrong.

Uncomfortable With Agile

Software project teams seem to show a tendency toward one of two extreme states:

1. A comfortable, repetitive routine (some may say “sonnambulant”) that ends in disaster when the team is blindsided by new developments (be they from the market, the technology, or the organization).

2. Chaotic disarray, where the main driving force is sheer panic, fueled by unrelenting overtime, conflicting goals and priorities, and enough poor decisions to fuel a Dilbert calendar.

Given the choice, however, you do not want to just aim for the middle of these cartoonish extremes.

Instead, you want to creep up to the very edge of that terrifying, wild mess of dysfunctional development disaster: up to the edge of the abyss, but without falling in. Think of that uncomfortable feeling of tipping back in your seat, just before you lose your balance and catch yourself. That is where an agile practice needs to be in order to thrive—balanced on a narrow edge that requires constant small adjustments to maintain equilibrium.

I think the best definition I have seen that captures this spirit of agile comes from Dr. Patricia Benner, author of From Novice to Expert. Speaking on the nature of expertise and how to train people in real-world practices (clinical nursing in this case), she said, “Practices can never be completely objectified or formalized because they must ever be worked out anew in particular relationships and in real time.” [2]

That is, you can never completely define agile, or its practices, because they are constantly evolving to meet specific needs in specific circumstances. Agile should be ever-shifting, ever-changing, ever-responding to change in context. As a practitioner of agile software development, you need to keep thinking, and keep adjusting.

I humbly suggest that this fundamental idea of agile development is what we (collectively) have forgotten.
What Is Agile?
In the crush and press of the exigencies of the day, we have forgotten some of the theoretical underpinnings of agile methods—why we do what we do. I would say agile includes ideas from:

- Chaos Theory
- Kaizen
- Systems Thinking
- Risk Management
- Return on Investment

Some of the key ideas from chaos theory include the extension of the pure physics version to organizations: that people, teams, and software products are inherently non-linear systems. That means that cause-and-effect can be impossible to isolate and identify. That is not a flaw in your team or project; it is merely a fact of life.

So is emergence, that nearly-magical phenomenon where complex systems and patterns spontaneously emerge from simple interactions. The converse seems true as well, that complex interactions tend to create simplistic, even “stupid”, behavior. This phenomenon is a large part of the agile drive to try and keep the interactions and rules of development simple, clear, and low-ceremony.

If you are an agile practitioner, do you regularly observe complex behavior arising from your code or your team? Or just the opposite?

Then of course there are the well-known ideas from Kaizen, emphasizing using quick feedback to make continuous changes, and so to create an environment of continuous improvement.

Are you using constant feedback to make continuous changes to your product and process?

From Systems Thinking, we came to the idea of people as First-Order Components of the systems we are building. We are not resources, we are not staffing, we are people, and we are as much a part of the system as the database is. We realized that it was up to us as individuals; that particular practices, special tools, or novel processes would not save us.

So are you investing in the people on your team? Buying them books? Sending them to conferences or training courses? Are you hosting brown-bag lunches to discuss technology and development practices?

Software development is risky business. There are many different ways to manage risk, and the agile way favors minimalism as a core approach. Timeboxing, whether in an iteration or in a design meeting, is a very effective way of minimizing risk. Another way of minimizing risk is to choose to write less code, or even better, not to write code. After all, the line of code not written is always correct.

But you already manage risk this way, right? You are constantly thinking about minimizing risk, are not you? You are timeboxing everything, from iterative deliveries and meetings to how long a team member can be stuck on a bug without asking for help?

Similarly, in addition to minimizing risk, you want to maximize business value—the stakeholders need to see a positive return on investment. That means delivering small bits of functionality frequently that help the organization achieve its goals.

Breaking Out of the Rut
Notice that each of these underpinnings of agile demands constant attention and thought. You cannot just run with a set of “standard” practices on autopilot.

You should find yourself thinking. Being skeptical. Re-evaluating. Are you doing some set of practices—are they working? Are you getting value for your effort? Is everyone on the team growing and advancing in their own individual careers?

In addition, there are definite warning signs to watch out for, which may indicate you are not working in an agile fashion [3]:

- **Sloganization**—Speech becomes so sloganized that it becomes trivial and eventually loses meaning entirely. Watch for all your favorite agile buzzwords and see if they are being used as meaningless jargon.
- **Confusion between following rules and exercising judgment**—When is it OK to break the rules? All the time? Never? Somewhere in between? How do you know? Some rules probably should never be broken (e.g., check everything into version control, have unit tests, settle on a fixed iteration length) and others may be more flexible (pairing conventions, specifics of customer involvement, etc.).
- **Confusing the model with reality**—A model is not reality. Thinking that an agile project is “supposed to go this way” might be a trap. The only thing it is supposed to do is succeed; everything else is up for grabs.
- **Demanding conformity**—The same standard may not always apply equally in all situations or with all people. You do not want a bunch of monkeys banging on typewriters to churn out code. You want thinking, responsible developers. Do not reward herd behavior or devalue individual creativity.
- **Spelling out too much detail too soon**—Premature optimization is indeed the root of all evil, whether it is in your process or your code. Tying things down too early is not agile, and excessive detail can act like instant glue. Have patience.
- **Oversimplification of complex situations**—“Just follow the process” If it were that easy, anyone could do it. But they cannot. Every project, every situation, is more complex than that. “All you need to do is…” or “Just do this…” are invitations to failure. Do not fall for it.

Become an Agile Practitioner
If any of these warning signs sound familiar, then you and your organization may be stuck in an un-agile rut. Here are some suggestions to help get things back on track.

First, the goal: you (personally) and your teammates (collectively) need to function as a true self-directed, self-correcting, agile machine.

Next, the problem: you are probably operating at an “advanced beginner” level of agile software development. No shame in that, statistically that is where the majority of agile developers will find themselves [3].

To understand this, let us take a look at the Dreyfus Model of Skill Acquisition, and then see how to move to a more advanced level of practice.
The Dreyfus Model

The Dreyfus Model [3] is a helpful way of looking at how people acquire and grow their skill sets. It defines five levels that you experience as you gain abilities at some specific skill.

Here are the levels, with some key features of each:

• Novice – Rules-based, just wants to accomplish the immediate goal
• Advanced beginner – Needs small, frequent rewards; big picture is confusing
• Competent – Can develop conceptual models, can troubleshoot
• Proficient – Driven to seek larger conceptual model, can self-correct
• Expert – Intuition-based, always seeking better methods

At the advanced beginner stage, you are comfortable enough to sort of muddle your way through. And for most folks, that seems good enough, so any advancement stops there. And unfortunately, this level is not sufficient for self-reflection and correction.

What we need to do is to see how to make the jump from an agile advanced beginner to becoming a competent agile practitioner. If you look at the levels above, one big difference between an advanced beginner and a competent practitioner is the competent’s ability to develop and use conceptual models.

So the first thing we need to do is to establish some kind of a mental model about agility.

An Agile Mental Model

Agile is all about taking small bites and focusing on continuous, not episodic, development. You want to do a little of everything all the time, and not face big lumps of major events. That is the "constant" part in this definition, "agile development uses feedback to make constant adjustments in a highly collaborative environment" [4].

The idea of correcting based on continuous feedback is key, but in order to do that successfully you will need to ensure that all the following are true:

1. All of your development and management practices generate continuous, meaningful feedback.
2. You are able to evaluate that feedback appropriately, in context.
3. You are able to change your code, your process, your team, and your interface to the rest of the organization and the customer in order to respond to the feedback.

That is just the mechanical portion. Any problems at this level have to be solved before moving on.

To be agile, you need this kind of base level environment, and then you need to apply an agile mindset. Look at the agile values from the manifesto: [5]
1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

If you can proceed in a fashion that honors these values and adheres to that definition, and you are able to respond as needed, then you are headed in a more-or-less agile direction. You have got the basic model of agility in your head, and by asking questions and constantly reviewing what you are doing in terms of this model, you will make progress.

Put it in Practice

That is the theory, anyway. But theory does you no good unless you can put it into your practice. And practice means trying, failing, learning, and trying again. There is no substitute.

Familiar and comfortable ways of doing things may look attractive, but that is a trap—do not fall for it. You will need to venture into the unfamiliar and uncomfortable, and lose a lot of mental baggage while you are at it. Think of the XP maxim and ask yourself, “What is the simplest thing that could possibly work?” What is the least you could possibly get away with? That question works for code, process, documentation, training, QA, and more.

Suppose you need to accomplish a particular goal. Ask yourself how can you accomplish that goal in a lightweight, low-ceremony, agile fashion, and still fully accomplish the goal? As an example, let us look at an aspect of testing that often causes confusion.

You have started a new agile-based project and have been getting everyone up to speed on this new “agile stuff.” Now you are tasked with bringing the QA staff into your fledgling environment to do testing. How can you do so in an agile fashion?

Let us clarify what it is they are doing—which is always a fine idea when starting out. We are not talking about “unit testing.” Unit testing is mis-named; it is a coding practice, not a testing practice. Yes, developers need to do it, and no, it does not negatively impact your QA staff. So perhaps your development team is doing test-driven or test-first development. That is great, they should be writing unit tests on an ongoing basis, keeping it a continuous practice and avoiding the episodic.

So now you need to bring QA into the fold to do more comprehensive testing, maybe focusing on acceptance testing, performance testing, other non-functional requirements, and so on. Are you going to wait for the end of a major release to give them code to test?

Of course not, that is thinking of the event as a large, discrete episode—that violates the model. You do not want episodic development, you want continuous development. Are you going to wait until the end of an iteration? Perhaps, but that is still waiting and making a bigger event out of the situation than is needed.

Give QA the ability to pick up a testable version right off your continuous built machine. Every build that passes all the automated developer tests is eligible, and QA can grab the latest functioning code base and have a play. They have the opportunity to grab releases every hour, every half-hour, all day long. Or they can wait and grab it once when done (in a non-agile fashion), but the choice is theirs and they can change that choice at will.

Does it Work?

How does this simple example stack up to our mental model? You have implemented a policy that is lightweight and easy to maintain. You have delivered value continuously, and you are delivering actual working software. QA can make their own decisions about when and what to test. You can experiment with feedback to the development team based on both team’s needs. So you are honoring individuals and interactions over some rigid process. You are able to respond to change easily, without affecting the development rhythm, as the two teams are loosely coupled (a great coding technique, works well with organizations as well).

Of course, things rarely go exactly as planned. That is just how the world is. Maybe this scheme does not quite work out for you. That is great! Evaluate the feedback on where the problems are cropping up, and make constant adjustments to try and fix it. That is what agile is supposed to be.

Just as with novice drivers, do not overcorrect, or you could slam into oncoming traffic (metaphorically speaking). Tweak a little, check the feedback, and tweak some more. Even corrections to code or process should not be big, major events. Change a little, all of the time.

And now you are practicing agility, the way we intended.

ABOUT THE AUTHOR

Andy Hunt is a programmer turned consultant, author and publisher. He authored the best-selling book “The Pragmatic Programmer” and six others, was one of the 17 founders of the Agile Alliance, and co-founded the Pragmatic Bookshelf, publishing award-winning and critically acclaimed books for software developers.

The Pragmatic Programmers, LLC
Website: www.PragProg.com
E-mail: andy@pragprog.com
Twitter: @PragmaticAndy
Blog: andy.pragprog.com

REFERENCES