Photo Credit: CollegeDegrees360 (https://creativecommons.org/licenses/by-sa/2.0) photo cropped/reesized

The Agile Manifesto - "Over" Does Not Mean "Ignore"

By Bob Bretall
Enterprise Agile Coach

When people start looking into Agile, one of the first things they usually come upon is the Manifesto for Agile Software Development. People, particularly developers, fall in love with the simplicity and economy of thought packed into so few words. It resonates with people who have been working in software because it just feels right. But the economy of the Manifesto is a both blessing and a curse:

Individuals and interactions ***over*** processes and tools
Working software ***over*** comprehensive documentation
Customer collaboration ***over*** contract negotiation
Responding to change ***over*** following a plan

The Manifesto emphasizes the points on the left, and rightly so. But in emphasizing the left, even by making the text larger so it stands out more, a number of people I have come across seem to read the core statements and take the word **"over"** to be a synonym for **"ignore"**. I have highlighted the word "over" above to emphasize its central place in each statement separating the left half from the right half. In seeing how some people react to the Manifesto and make statements about Agile on message groups and forums, it would seem that they didn't fully understand next statement that ties it all together:

That is, *while there is value in the items on the right*, we value the items on the left more.

THE DESARA GROUP

We don't want to just throw out the items on the right if we really want to implement an Agile development environment instead of just "iterative hacking."

Let that sink in: **There is value in the items on the right** (*just not as much as the items on the left*). We really need to emphasize the items on the left because they are things that sometimes got overlooked or ignored in past high-ceremony processes driven more by business rules and protocol than by engineering need. We don't want to just throw out the items on the right if we really want to implement an Agile development environment instead of just "iterative hacking", which is what some Agile implementations devolve into.

Individuals and interactions over processes and tools

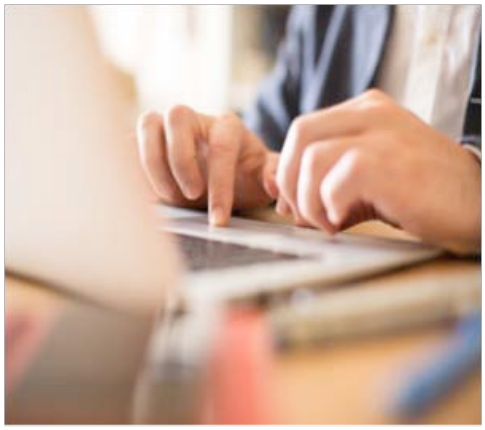
We absolutely need to focus on the people doing the work. We need to make their lives easier; removing obstacles that stand in the way of efficiently developing code and allowing them to interact both with one another and cross-functionally with others in the organization that contribute to creating a working product that does what it needs to do when it needs to do it. This is most important and correctly emphasized.

While we should not be driven by process and tools, process and tools can definitely play a key role in assisting individuals create quality software.

Let's start with tools. Most people don't take this statement to mean "don't have/use any tools". They quite logically see the value in many tools that augment software development and delivery. We benefit greatly from tools that augment our ability to both create software as well as those that enable enhanced interaction among team members. We don't use tools that don't add value, we use those that have a specific purpose and add value.

It's interesting to me that while people understand that we should use only those tools that add value, they often take the other part of the statement to mean "throw out all process". Quite possibly part of some in the Agile community considering ***process*** to be a dirty word. We need process just like we need tools; what we don't need is process that does not add value. Don't have approvals and gates that exist to slow down the creation of working software. Don't have giant checklists that seem to serve no function other than to be filled out and then looked at by someone somewhere down the line that has nothing to do with creating/delivering the software product. Don't have process steps that don't add value. Don't slavishly follow and document process steps just for the sake of doing the steps. Anything we do should be done because it adds value and we should make sure the team understands the value those things add.

Process and tools need to be right-sized. They should add value, and we should all understand the value they add, or they should be excised from the development environment.



Documentation is needed any time we know someone other than the original developers are going to need to understand, modify, or maintain the code.

Working software over comprehensive documentation

Absolutely! The core of Agile is having working software created as an outcome of every iteration/sprint. Nothing should stand in the way of this core tenet of Agile.

But look at the right side of that statement: it says **comprehensive** documentation. It does not mean to say that we should create **NO** documentation whatsoever, though that is how some interpret it.

What we should **not** do is divert development team effort into creating huge forest killing documents that nobody will ever read and which are created to check off a box somewhere that we have documented our software system.

Eventually people will come along and be added to the team who will need to understand what has gone on before they got there. While we can bring those people up to speed via a live mentoring process, what if the product moves past initial development into maintenance mode and there isn't someone around to recount tribal knowledge while sitting around a microwaved burrito in the lunchroom?

What we need to do is make sure that the documentation that naturally occurs as part of our development process is filling the need for documentation for people that want to understand the system. We also need to consider if there are requirements for documentation that are necessary in order to comply with an external standard, in which case we need to do a mapping of what is required to what we are actually producing. Making sure we have documentation sufficient to pass a regulatory audit is a separate issue and if needed, must be addressed as part of our development process.

As with process and tools, we need to right-size documentation. This is done by understanding what documentation needs to be used for down the line, particularly if maintenance or regulatory compliance is in the future. Documentation is needed any time we know someone other than the original developers are going to need to understand, modify, or maintain the code. Then we need to ensure we plan to spin the naturally occurring documentation into a form that will meet these needs as efficiently as possible.

Customer collaboration over contract negotiation

We should aspire to work closely with our customers in order to make sure we're building what the customer wants. The worst thing is finding out too late that what we (or the customer) **thought** they wanted is not the same as what they really wanted or what they actually needed.

The dig at contract negotiation is a call-back to a mentality where we negotiate a price and a set of requirements, negotiate a price associated with that, and then go through an additional set of negotiations for any and all changes that divert from the agreed upon contractual obligation. If you've ever had to do contract negotiations (I have, and it's really not fun) you realize that when the contracts people are involved, everything moves at a glacial pace and is pretty much the antithesis of agility. If we are in an environment where we need to go through contracts for any change you can guarantee that change will take a long time and will be a fairly lengthy process.

But in most cases we cannot throw out contracts altogether, since we're still operating in a business world that exists in the context of contracts, lawyers, etc. We need to have some kind of contract, but we need to make sure we don't let the contract get out of hand in squelching the agility of the development process which is geared towards quickly and regularly delivering results of value.

We need to be able to collaborate with the customer in ways that add value without being bogged down by negotiating new contractual clauses every time we need to make adjustments along the way to delivering something that is best for both the customer and us. Think about when a contract actually gets pulled out in order for someone to enforce a contractual clause. It happens when someone is dissatisfied, either with the way they think things are going along the way or with an end result. Regular collaboration, real conversations/communication, and involving your customer in decisions along the way will keep them aware of how things are going, keep them happy, and generally ensure they are not pulling out the contract to "enforce" it.

Are you starting to see a theme here? Contracts need to be right-sized to lay out general terms and we need to make sure we deliver on that with a quality product that we and the customer are both happy with. No surprises; they will get what they have been seeing evolve from the beginning and were involved with all along the way.

Responding to change over following a plan

How would anyone NOT do this? The essence of Agile is being responsive to change. It's built to be responsive to change. Should we build a really detailed plan that extends months out into the future that is guaranteed to change before we get to the iterations described by most of it? Of course not! Though a number of companies still do this, in my experience, the need to have a detailed long-term plan is an exercise of false accuracy that is often fueled by parts of the company outside the engineering/development organization (for instance finance requesting detailed forward plans for budget planning).

Does being responsive to change mean having NO plan? Of course not! You can accurately plan details for near term tasks, so we exercise Just-In-Time planning at the detailed level. That said there should absolutely be a coarse plan laid out that takes a long-term view into account. This can be easily adjusted as needed since it should not be something that has required a tremendous investment of time. But it gives us a high level idea of not only what our end point is but some of the major milestones along the way, all of which can easily change when needed.

Plans need to be articulated in a level of detail that is reasonable assured to be as accurate as possible given the nearness of it's timeframe to when the plan is created and takes into account both near-term and long-term priorities and goals. Since the team is involved in planning (it's not just done off in a separate room by a project manager and then handed to the team) it is a great idea to do an initial high-level plan so the team can understand where they're headed and not get tunnel vision in seeing only the extremely near-term goals.

We need to take into account that there are sometimes audiences for plans outside the immediate development team. We can educate people outside the team to use the plans that make the most sense for the team, but there should be some kind of tangible plan that articulates what is being done and when.

We need plans and we need the right level of detail in our plans at the right time.

Does being responsive to change mean having NO plan? Of course not!

Negative views of Agile come from those implementations that totally dismiss the things on the right side of the Manifesto.

Right side = Right-sized

As you may have noticed, it's all about right-sizing the things on the right in The Agile Manifesto.

A lot of the criticism I see leveled against Agile has to do with the tendency of some people implementing Agile to participate in some form of "iterative hacking" or "cowboy programming". The criticism is a reaction to problems caused by not really taking advantage of all the benefits Agile can bring. Problems can be brought on by lack of planning, inconsistent application of sound software engineering principles, hard to maintain code, and lack of delivering an end product that actually meets customer need on time and within budget.

Negative views of Agile come from those implementations that totally dismiss the things on the right side of the Manifesto. Get the results you need and have more positive outcomes in the long-term by giving appropriate attention to both the left and right sides of the equation. You'll be glad you did.

THE DESARA GROUP

Bob Bretall

Enterprise Agile Coach

Bob Bretall has spent over 30 years in the software development field with hands-on experience in all aspects of the software development lifecycle. He has been a developer, designer, team lead, manager, trainer, mentor and consultant.

Read More **Bob.Bretall@DesaraGroup.com**