YOU AREN'T AGILE

Linda Wilkinson

Wilkinsonlinda4@gmail.com

Abstract

Agile methodology is 24 years old. It has a manifesto. It has underlying principles. Do you know what those are? Support them? Believe in them? Many companies assume Agile is just "whatever works". It isn't. The resulting odd mish-mash may or may not be working for you. At the very least, every person you hire will have their own ideas of what Agile means based on their own experiences and it takes time to figure out what it means in their new organization. The problem is worse when you add a new manager, director, or VP to the mix. They will inevitably want to impose their own concept of Agile in their new organization. Again, this is a time-waster for tech.

The purpose of this highly-interactive presentation is challenge you, have some fun, give out some swag, and help all of us, as a field, to evolve in directions that make sense.

Are you Agile? I think not...

Prove me wrong.

Biography

Linda Wilkinson has been a QA practitioner for 40 years (!). She started her career in software analysis, moved into QA, and followed the traditional route of QA Analyst I, II, II, Senior, SDET, Lead, Manager, Director, VP, and took a few brief forays into DevOps, DBA, Product Management, and data analysis as it became necessary in order to accomplish quality goals. At last count, she has worked with 18 languages, taken hundreds of courses, and has had the privilege of speaking at a variety of conferences, both as a speaker and as a member of expert boards in QA and metrics. She has built QA teams (functional, automated, and performance) from scratch five times and recently published a book about her lifetime in IT, Rhinestone Quality. Passionate about quality and providing business value, her teams have created and maintained error rates close to 1% with high customer satisfaction ratings in a highly competitive and ever-changing technical landscape.

1 Introduction

Do you work with or for a company that proudly proclaims it is an Agile Shop? Are you really agile? Have you ever actually read the Agile Manifesto or its underlying principles? If not, how do you know? Let's take a look at those things.

The original Agile Manifesto was relatively simple:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

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

That's it. It first came out in 2001 and was a big deal in the IT world at the time. Everyone was lining up to sign the manifesto and "big names" in the field showed off their signing and support. The entire effort was based on a small, tight team that pulled off a major effort by themselves and who later documented what they had learned and attributed their success to the adoption of some repeatable processes. Later, when they had been inundated with questions and requests for more information, the manifesto was expanded to include more detail regarding the principles behind their manifesto. I'm including all of this here because it's a rhinestone philosophy when you say you do something you clearly don't do. These are the principles they follow today:

1. **Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.**
2. **Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.**
3. **Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.**
4. **Business people and developers must work together daily throughout the project.**
5. **Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.**
6. **The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.**
7. **Working software is the primary measure of progress.**
8. **Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.**
9. **Continuous attention to technical excellence and good design enhances agility.**
10. **Simplicity – the art of maximizing the amount work not done – is essential.**
11. **The best architectures, requirements, and designs emerge from self- organizing teams.**
12. **At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.**

This methodology is not new – it's 24 years old. On the surface, all of the above sounds great. In practice, not so much.

I never signed the Agile Manifesto. At the time it became visible, I was working in some highly regulated, heavily audited organizations which would have failed if they had followed these principles. To be completely honest, Agile doesn't mean any more (or less) to me than Waterfall, RUP, or any other software development methodology. There are always some parts of any given methodology that do (or do not) work for your particular company.

In other words, I don't care (!).

If I wrote my own manifesto, is would be:

"A great team will survive, surpass, and succeed regardless of methodology and obstacles.".

That's it. Feel free to sign up. But back to Agile, and why you aren't. Maybe you wish you were agile. You kinda are – after all, you hold retrospectives, don't you? However, when you say you are something you're not, you are lending your credibility and support to something that not only does not work for you, but that likely doesn't work for many companies and actually holds the field back and stops it from evolving. Worse, you're doing a disservice to your own company. From a corporate perspective, it isn't especially helping your bottom line, which is dependent on producing awesome things people want to buy as quickly and efficiently as possible, by wasting time and money on processes that are not of fiscal benefit to anyone. If you have a chaotic, messy process that burns people out, doesn't meet deadlines, adds tons of tech debt to your backlog, and works like anything but a well-oiled machine, you're not agile.

Everyone says they're agile, so you aren't alone. Everyone wants to be agile. It's the word itself. Who wouldn't want to be quick and well-coordinated?

But chances are good you don't really follow agile principles. Maybe, once you think about it, you really don't want to follow agile principles. Let's look at why.

ARE YOU AGILE?

Individuals and interactions over processes and tools

If you are committed to AI or automation in terms of the bulk of your efforts in order to be faster and cut down on the number of people you hire or retain, you value processes and tools, which you likely equate to time and money and you do not value individuals and interactions above those things. This is a natural tendency for technologists. Engineers grow up to be managers, directors, VPs, CTOs, etc. Talking about tools and processes push all their "good" buttons. It's what they've done, what they do and understand, and what makes them comfortable. They'll bring the rest of the executive team along for the ride when they mention time and money, whether that is actually true or not and regardless of the end result to their customer base. If you operate this way, and many do, it means you are already at odds with 25% of the founding principles behind agile methodology.

Working software over comprehensive documentation

What do you mean by "working"? If it is something actually made available to and useful to a customer, many agile efforts fail to do that. If it means a piece of software that appears to work by itself in a virtual vacuum and which will be unavailable to the customer until the feature or application is more complete, that makes more sense. And what is "comprehensive"? I'm not a fan of red tape and I'm sure you're not either. If comprehensive means "anything more than absolutely necessary", I think we can all get behind this. But even "necessary" is going to differ tremendously between companies depending on the nature of their business. A highly regulated and audited field requires more documentation than a start-up in other fields. There are also companies that have political climates that are so vile they require a lot of red

tape and sign-offs just to force people to work together and get basic work done. So overall, the concept of working software over comprehensive documentation is vague. We get the idea, but these are weasel words if we have no definition.

To put a more positive spin on it, however, you can define this yourselves. It's your company and your environment. What is "working software" to you? What is enough documentation? What is overkill? And why even waste time talking about it? Because if you don't, every person you hire is going to have their own interpretation or if in a management position will try to impose their own understanding on you – wasting your time (and of course, your money). Being efficient as a company means being self-aware and cutting the crap, making sure everyone understands and is on board with how you do things and want to operate and what you value. If you have upper management, laying down strategic guidelines is part of their job. So get strategic. Otherwise you'll be trying to manage a free-for-all. That can be fun, but it's also exhausting and not sustainable long-term.

Customer collaboration over contract negotiation

I wish this said "customer collaboration is as important as contract negotiation.". If your company does business through contracts, there are too many things critical to your success as a company to take a backseat to anything. Poorly negotiated contracts can bankrupt you or get you sued. Bad contracts inevitably result in cost overruns and a loss of revenue. You can still be committed to customer collaboration without sacrificing contract negotiation. If you believe this as well, the concept represents 25% of the core values behind agile methodology.

Are you still with me?

Responding to change over following a plan

Wow. Many "plans" in terms of agile methodologies are a few weeks long and consist of pulling a number of tickets in front of the team to be "worked". There is no real "plan". Significant, constant change creates chaos and ultimately translates into lateness on other things – some important and some less so. Constant change is the bane of many agile teams' existence. Their shops are chaotic, messy, and nothing ever seems to get done, de-motivating and frustrating many of the staff. Change needs to be intelligently managed so it doesn't negatively impact your progress or your teams. Every change should be evaluated and if it isn't critical, it needs to go to a later sprint (or phase) in order to protect the team and their progress. This statement, given the lack of plans inherent in today's technoscape, might be better phrased "Responsive to change".

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

I laugh every time I read that – it's such a vague, steaming load of excrement. Does your customer want continuous delivery of software? Is it actually valuable to them, or valuable to you? How do you measure whether you satisfied your customer? This should read "Our highest priority is to increase our bottom line through early and continuous delivery of software that makes us more money.". First, people as a whole do not respond well to change. It is unlikely anyone wants software that is constantly being changed, unless the change is a fix. In some cases, constant change can actually drive people away. You need to know your customer base and what is and is not acceptable to them.

Many companies deal with "early and continuous" by actually hiding new features and functions behind something like Split.io until a nice, marketed upgrade is scheduled with a bit of fanfare. Some make the upgrades available gradually, a few clients at a time. Regardless, the real driving force here is not the

customer, except as a limitation. The driving force is an ever-increasing, ever-improving product base that delights your customers and thus increases your market share and value as a company. It is much more likely your company's highest priority is to make more money for the company. Companies, in general, are not altruistic. People are altruistic.

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

Do you know anyone who welcomes changing requirements, especially late in development? I've been doing this for 40 years, and I have to say I don't know one. Agile processes do not harness change for the customer's competitive advantage. There may be no competitive advantage whatsoever for the customer.
It's more likely the change is advantageous to the company's competitive advantage. And if it isn't, why are you doing it? Unless it is something being demanded by some of your largest and most influential customers, it is likely something that can be deferred to a later sprint, when it isn't disrupting your progress and your teams. Generally speaking, it is a mistake to allow the tail to wag the dog. What a short agile process does allow you to do is to incorporate change more quickly – you can choose to place changes in later sprints without blowing what everyone is working on out of the water.

I once worked for a company that was virtually hamstrung by dropping everything to work requests from their largest customers. They ended up being used as personal IT departments by these customers and were making no progress against corporate goals that involved updates and new features for all of their customers. Eventually, operating this way could not be sustained – they had to stop. They lost some of those customers. If they had set boundaries earlier, it would have been less of a problem. This is especially prevalent with small companies and start-ups, who are more dependent on large, vocal accounts.

Good teams can and do incorporate change, but will avoid disrupting their work whenever possible. Ultimately, what you want is up to you. Is a chaotic madhouse scrambling to get things done at the last minute every two weeks attractive to you? It's certainly exciting. On the other hand, it's certainly the opposite of a well-oiled machine and there's a lot of burn-out. Human beings cannot maintain that level of excitement and terror long-term and after a year of that craziness, with the accompanying overtime and angst, your staff will cease to be energized and will no longer pull together to get things done. When everything is an emergency, eventually nothing is an emergency.

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

I can only assume "working software" means a piece of code that appears to work by itself and that will be hidden from and not usable by a customer until the entire feature or function is done. There is a huge difference between providing a fix and providing a new feature/function. Fixes can and should be made available as they are ready. New features, functions, or applications are marketing opportunities that you will likely want to introduce to your existing and potential customer base in a way most beneficial to your company and easiest for your customer base to accept and even look forward to, thus making you maximum money on your investment.

New features and functions might be extremely simple – something within the capabilities of a single team in a single sprint (or block of time). But a new application or some features will be bigger than a breadbox and will require multiple sprints to complete.

I've worked on efforts managed through agile methodology that took two years to complete. Progress was still measured in two week increments, but the final product was not complete and offered to customers for two years. Take a look at that. Two years sounds waterfall-y, doesn't it? But it isn't. One

thing agile is really, really good at is sweeping their failures to get work done in a timely manner under the rug. If something isn't done on time, it's moved into the next sprint/block of time. This isn't necessarily a bad thing. Essentially, development is a creative process. How long does it take to catch a fish? Paint a picture? Things come up that were unplanned, things turn out to be more complicated than anticipated, in essence, shit happens. Agile methodologies are better at hiding how long it really takes to get something meaningful to the customer. I'm not sure whether that's a benefit or a fault. You decide.

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

If "business people" is equivalent to the product owner, this can happen. If it means anyone else, it won't. Business people might be involved during the ideation and specification phases, but that will likely be it until there's a product out in the market. Often customers aren't involved at all. Look at your own company and situation. Do you have business people, developers, and maybe customers working together every day? If so, do you find that type of collaboration speeding up your process? Do you want to add this process? Even if it slows down your progress?

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

There is nothing a motivated, talented, tight team cannot accomplish. Nothing. But this is genuinely one of the hardest of all the principles to get right. First, do you have any teams you can definitively say are motivated? Or have you sucked the life out of all your talent? Or not hired the right talent? Have you built out the kind of environments they need to do their best work? Do you support them? Do you even understand what makes those team members tick and do their best work? If not, you're not agile.

Regardless of project methodology, this principle is surprisingly difficult to get right.

One of the key points in the agile principle is "trust them to get the job done". Trust is an unbelievably hard thing to come by in most companies. When you read the many complaints out there about managers, much of it stems from lack of trust and micromanagement. Hire the right people, give them what they need, and then get the hell out of the way and let them get the work done. A really good team doesn't need you in order to do their thing. The team needs you to do things they can't do – and that shouldn't be their day-to-day work. If they are programmers, you shouldn't need to look at their code to make sure they're programming things correctly. If they're QA people, they don't need you to check their test automation or test cases. You're going to know if things are going awry by results, at which time you can step in and course-correct if necessary. Pair senior people or leads with those less experienced and let them gain experience with training and leading teams, as well as counting on them to lead through example. Encourage your people to step up. Train your people to step up. Believe in their ability to step up.

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

Now be honest. Do you have co-located teams? Or do you have remote teams or some members that are remote? Then by definition, you cannot be agile. In spite of the fact that many of you will hate me for saying this, I agree with face-to-face conversation as the most effective means of conveying information. I also believe teams are more creative and innovative when they're collaborating with each other in person. But if you don't support that (or can't support that), you aren't really following agile precepts.

Agile methodology supports working software as the primary measure of progress.

Every software development methodology supports the same thing. The only caveat with agile methodology is what you define as “working”.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Nothing I’ve ever experienced in regards to companies that say they are agile shops supports this. Most companies choose Scrum (a type of agile methodology) to produce software and most choose two-week “sprints” (blocks of time in which to produce results). The pace that has to be maintained tends to de-motivate and burn out staff, unless the only work being done are fixes or tech debt. Developers tend to be optimistic about what they can accomplish in two weeks. That means they have to push, really push, to meet their commitments in a two-week timeframe, often with overtime and a lot of urgency and anxiety around the deadline. They go through that every two weeks with Scrum. It is not sustainable.

Most human beings and teams do better work with some sort of deadline. It’s a goal that keeps them moving forward at a good pace. But two weeks is usually an arbitrary and meaningless number; this is usually just pulling a date out of the air for no reason whatsoever. Why would you expect good results from this?

Continuous attention to technical excellence and good design enhances agility.

That is an absolute lie. Technical excellence and good design take more time. It also takes more training. I strongly believe this is part of agile methodology principles because they were written by and largely for technologists and all of them want to incorporate this into what they produce. Excellence, in anything, requires more time, money, and commitment than mediocrity. Do you have a lot of tech debt? Things you should have done but didn’t have time to do? Then you don’t adhere to this principle. Here’s the painful part. Maybe you don’t need to in order to be successful. When you think about this, you realize there are many products out on the market, making money, that are neither technically excellent or lessons in good design. They may, however, fill a niche and be valuable regardless.

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

We can likely all agree this is a good idea. Extremely difficult for technologists, though. Overdesign was, is, and will continue to be an issue for technologists. In today’s technoscape, many customer-facing applications are designed by non-technologists who know or collaborate with your customer base. Customer-facing products designed by technologists need strong vetting prior to releasing them to production, or you may end up with a website 12 pages long that your user base abandons after page 3. Fortunately, anyone outside your engineering organizations can “beta” your product to ensure it is fit for use.

Keeping things simple is not really a new or strictly agile concept. If you take a look at what we actually have to work with out in the wired world, it’s obvious that “simplicity” isn’t necessarily a priority. In addition, consider how companies really work. You may design a product or a suite of products. Your customers love your products. They are the highest-rated products of their type in the world. And yet. And yet. You keep poking at them. Changing them. Tweaking them. Why? Your customers already love them. You’re already #1. Is it because you have all these technologists on hand with nothing better

to do? You have nothing better than to spend your money and time fixing something that isn't broken? This is where you can genuinely surge ahead in the field. Spend that time and money on new products. Innovation. Marketing. Anything but messing with something that is already working for you. Most companies indulge in this type of wasteful and largely useless activity. You don't have to be one of them.

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

There are a few agile principles that sound good but are, in practice, really dangerous. This is right up there. This principle simply assumes too much. What makes you think anyone on a given development team can organize as much as their own sock drawer, let alone a team? There are brilliant technologists out there that have difficulties tying their own shoes.

Architectures are usually applied across many teams, not just one, and they don't magically emerge from untrained or inexperienced people. Also, unless your teams include customers, and most do not, what makes you think a development team, which will contain people that hate to document anything, are going to produce the best requirements? What if you have a team that is primarily noobs? Do you really think they will produce the best designs? Have any of them been trained on design? Do you know? The best architectures, requirements, and designs emerge from teams that have the right people in the right roles, organized and run by someone who is experienced and in collaboration with other teams that have a stake in the game.

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

Every agile team I've ever worked with held retrospectives. Be honest. Do they actually change anything based on those retrospectives? If they produce something valuable to you, great. If, like me, they have always been a well-meaning effort that never resulted in any valuable changes.

2 CONCLUSION

You now have a complete picture of what "agile" is all about. So are you truly agile? Do you want to be? The Agile Manifesto and associated principles might be something you want to follow, but you need some help. Check out the references to get started. Agile can be, and has been, extremely successful for some companies.

But maybe you and your company have evolved beyond the original ideas offered by Agile methodology. So what's next?

Why don't we redefine our methodologies to better reflect what "fast and well-co-ordinated" really means? First, you don't have to do things because those 200 companies over there do things that way. Those companies are not your company. You can forge your own path, define your own processes, and those processes can help your company succeed where others, who then become merely followers, continue to waste their money and their time allegedly conforming to something they'll never conform to. Your process may not be "THE Agile Process". Instead it can be "The (Insert your name here) Company Process". Another thing to mull over is to create the next Great Thing. It's long overdue for a new methodology that benefits the majority and our current technoscape. Maybe our next Great Thing should incorporate business operations in a world that has changed considerably over the past 24 years. Just remember, T-Rex was agile in his day too.

Clinging to something that doesn't work for your company and will never work for your company is not the way to go. Rather than addressing and solving your problems, it reinforces and contributes to bad behaviors and lost revenue. You're just laying a generic happy word across the messy chaos that is your actual existence and calling it handled. It's so much easier to do it that way. Maybe so. It's also much more expensive. Strategize. Innovate.

How do you start? How do you strategize any problem areas? Work with your peers, your employees, and your bosses and find out what does and does not work for them. Take those things that do not work and collaboratively brainstorm solutions to those problems - strategizing ways to fix what does not work. Experiment with those solutions until you find answers that work for your specific organization. Communicate it. Create a methodology that works for you. Move yourself and your company forward and let the rest of the world catch up to you!

References

agilemanifesto.org

agilealliance.org