# Agile Manifesto — Back to the Basics

At the beginning of this millennium, I was working in consulting at a company that paved the way when it came to modern systems development and methodology. During this time, we preached this excellence through iterative methodology. We talked a lot about Rational Rose and Rup, and our customers both listened and implemented. And it led to good results.

Then came five years at Microsoft, where I didn't work on development projects at all. Nor did I pay attention to what was happening on that side of the industry. So it was a big surprise when I made my way back to consulting. The methods had developed immensely, and there was a great emphasis on productivity. Methods had grown more detailed, more formalised. But was it any better?

In basic terms my job involves hiring out system developers and -architects. And that was what the customers hired them as. But time and time again they were sent to help the customers get Scrum working. A great many of them had decided to get agile, and Scrum was the natural solution. They jumped right on that method and implemented without fully understanding why they were doing it, nor what they wanted to achieve.

For a while I wondered if I had misunderstood the word "Agile", since all I saw was anything but my idea of agile. And when I read the Agile Manifesto I had even more doubts. There didn't seem to be any clear connection between the thoughts of Agile and the so-called agile methods.

Here are the principles of the Agile Manifesto:

- **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 sounds reasonable, doesn't it?

In addition to sounding reasonable, a number of scientific studies indicate that the writers of the Agile Manifesto really were onto something clever. A few years ago, MIT published the results of a major study on what it is that makes some teams more effective than others. More than 2000 people participated in the study, and its subjects were teams across different industries, as well as teams in the same businesses. The results were striking. The factor that made the biggest difference to the quality of their results, was the way in which they communicated. It was more important than every single other factor added together: age, intelligence, education, etc.!

The characteristics of the most successful teams were:

- Everyone in the team spoke and listened in mostly equal measure, keeping their contributions short and concise.
- The members looked at the person they were speaking with, and they had engaged conversations.
- The members brought up topics with each another directly, not just with their leader.
- The members talked amongst themselves, even outside of meetings.
- The members sought information and inspiration outside the team.

A lot of this overlaps with the points raised in the agile manifesto, right?

Considering how obsessed our industry is with being agile, I think its peculiar how people have been able to conceal the core of the Agile Manifesto, to the extent that they ignore it entirely when they want to be agile. Because what is it, to speak plainly, that actually happens when a company decides to work agile:

- **Individuals and interactions** over processes and tools
    - We'll implement one method and a ton of tools we don't fully understand.
- **Working software** over comprehensive documentation
    - We must minimise the number of bugs. Whether that actually solves the challenge in question isn't too important.
- **Customer collaboration** over contract negotiation
    - Agile is something we work on in the development department.
- **Responding to change** over following a plan
    - The most common thing is to make what you guess the users need, and to continue guessing until the users are so sick of it that they don't bother, or until you're lucky and guess correctly. I call this Presumptive Development (PD!).

I'd just like to stress that I don't mean to say that it's wrong to use Scrum or other methods and tools. I just think you need to have something else in place first. Something like a foundation for these methods. Without a solid foundation, you can't build a house that will stand the test of time. So what do I think should be done? Let me answer that point by point.

## INDIVIDUALS AND INTERACTIONS - ***People and relationships***

Instead of starting by implementing Scrum, how about focusing on the people in the teams and their relationships? A lot of people think relationships are built by going out into the woods to challenge obstacle courses or making plays together. And suuuure: you get to know each other. But in a very situational way.

Instead, how about sitting down and letting people talk about themselves a little? Who they are, where they're from, their families, their interests, what motivates them? I've done this exercise many times. It demands very little, aside from the willingness to share a little about yourself. And I've never seen the participants fail to learn quite a lot about each other. Even with people who've worked together for many years.

Instead of taking a cooking course, how about consciously developing the way in which the team members listen and talk to each other? Are they doing their best to understand one another, or are they just sitting there, listening for mistakes? Are they trying to make good solutions through impassioned dialogue, or is every single one of them trying to win the discussion? Good interaction requires both conscious involvement and practice.

## CUSTOMER COLLABORATION - ***Trust and cooperation***

Does the team consist of all the people and all the knowledge you need to create a good solution? Are the users and the business end represented in the team? Is it dedicated to the final product? Are the people who will operate the solution involved in the project? Have we done anything to build trust within the team, as well as trust between the team and the project owner? Do the people who are paying for your solution trust the people making it?

## WORKING SOFTWARE - ***Transferring knowledge, dialogue and communication***

In order to create a good solution, one has to transfer knowledge from those who understand the domain and the business to those who are making the solution. The ones making the solution, in fact, need to understand the domain even better than the ones using the solution. After all, they have to make something that works better than what is being used today.

To effectively transfer knowledge, we first and foremost need to have the right people involved in the project. They don't need to be 100% allocated, but they must be accessible to the other participants. Both in terms of physically sitting near the other team members, and by keeping some time open every day for questions and discussion.

Afterwards, one must strive to ensure that everyone in the project is speaking a language the others can understand. The business side must be able to explain the challenges in a manner that can be understood by non-experts, and the technologists must learn to listen and ask questions. Far too often people enter a compilation mode and listen for mistakes and logical fallacies. Yes, at some point you DO have to do this as a developer. But maybe this can wait until you've settled into the central issue? One must first understand the nature of the problem and decide if something should be done about it, before one can start thinking of a solution.

Another often-ignored factor is the complexity of organisation and its solutions. Often there isn't anyone with a full overview of how everything fits together. Gaining an overview of both the business processes and their associated IT solutions is an extremely useful but highly underrated exercise. This can help both the businesspeople understand the IT solutions better, and the IT people understand business better.

## RESPONDING TO CHANGE - ***Evaluation and Evolution***

The best way to respond to change is to be proactive. By producing suggestions for improvements and adjustments as soon as possible. To achieve this, one must regularly evaluate the work that has been performed so far as well as the manner in which it has been done. One must at all times keep an eye out for the small things that can be improved.

In order to succeed at this, one must ensure that everyone working in the team cares about their work and the results that are being achieved. They must trust each other enough that they dare to be honest. They must have good enough relationships that they dare to be critical, even of each other.

Sometimes the need for changes is unexpected and far-reaching, and it is at this point that one is dependent on a team that is able to work together and carry out the changes. But in our world there are also other frames that allow us to implement changes. For example, the code quality and architecture of the solution. Have we remembered to cut the technical debt, so that it's actually possible to make code changes?

Do you think this sounds exciting, but that the blog post isn't enough for you to do something about it? Don't despair! We've made a course that can be adapted for developers, project leaders and management.

At the course we'll discuss:

- What are the core concepts behind Agile?
- What characterises teams that deliver extraordinary results? How does this correspond with the Agile Manifesto?
- What can be done to make you more agile and achieve greater results? We'll give you an introduction to the theory behind it, and a set of tasks and exercises that will help both you and your organisation on your way.
- What framework is needed to succeed at working better together? Not everything can be solved by Agile. What other frameworks should be in place?

If you're interested, please get in touch at kjell@webstep.no