How to Shoot Yourself in the Foot. In an Agile Way

Giovanni Asproni, gasproni@asprotunity.com

Introduction

The project is late and over-budget, the software is bug-ridden and unusable, the customer is furious and doesn't want to pay any more money, the team is burned out and lacks motivation. The Project Manager, looking around for advice, comes across the Agile Alliance web-site [2] and decides that an agile methodology is the way to go to rescue his project…

This is a typical scenario of introduction of an agile methodology in a company; of course it is not the only one—some projects use an agile methodology right from the start. However, no matter how and why an agile approach is chosen, there are some traps and pitfalls that it's better to be aware of.

In this article I'll describe what, in my experience, are the five most common and dangerous mistakes that can make the implementation of an agile methodology fail. I'll give also some hints about how to avoid and/or fix them.

In the rest of the article, I'll refer to the person that has the ultimate decision on what the software should do as the *customer*, and to the developers as the *team*.

Finally, the *project manager* is the person that, in a traditional process, is in charge of planning the activities of the team and the deliverables of the project, and, in an agile one, works more as a facilitator between the team and the customer, and makes sure that the activities of the team run as smoothly as possible. In both cases, she also keeps track of progress made and is instrumental in keeping all the stakeholders focused on the project goals.

So, you want to be agile

Nowadays, agile methodologies are in the mainstream, and almost everybody claims to be agile: every time I talk about agile development with some project managers or developers, the first point they frequently make is "in a certain way we are agile as well". (OK, sometimes they are not really that agile, but this is something for another article).

I personally believe that agile methods can give many projects a boost in every respect: quality, productivity, motivation, etc. However, their adoption may be more difficult than many expect. The reason is that, in every agile methodology "Individuals and interactions" are considered more important than "processes and

tools" [1], and managing people it is arguably one of the most challenging activities in every organization.

There are many ways to make a project fail, but, in this article, I'll focus on what in my experience are the five most common (and dangerous, since any of them can make a project fail) mistakes that can be made in the adoption of an agile methodology:

- Mandating the methodology from above
- Lack of trust
- Measuring agility by the number of "agile practices" implemented
- Thinking that merely implementing some of the practices will improve quality
- Focusing too much on the process and not enough on the product

Let's have a look at these mistakes in more detail.

Mandating the methodology from above

This happens when the project manager (or some other manager) decides that the team must use an agile methodology in order to be more productive, and imposes it on the developers (and sometimes, on the customer as well).

If the project manager is lucky, and the team members already think that an agile methodology is the way to go, this approach might actually work. Unfortunately, imposition very rarely works with programmers, especially with the good ones: programmers are knowledge workers and, as such, they don't like to be patronized, especially about how to do their job properly. So trying to impose them a new methodology can actually have an effect opposite to that which was intended.

I worked in a project where Extreme Programming was imposed from above, and the developers where forced to pair program all the time (with even the pairs often chosen by the project manager), no matter what they thought about the practice, and, as a result, some of the programmers where quite grumpy during pairing sessions making them very unpleasant. Later in the project, after having seen the effects of his decision, the project manager changed his mind, and made pairing optional leaving to the programmers also the choice of whom to pair with. Suddenly something interesting happened: the programmers that used to hate pairing chose to pair most of the time; the only difference was that now they had freedom of choice and decided to choose

what they thought was best for their productivity.

If you are a manager willing to adopt an agile methodology in your company and also want to succeed doing it, you should consider involving the programmers and the other stakeholders right from the start, asking for their opinion and help. You may also be willing to consider the books by Linda Rising, and Mary Lynn Manns [3], and Jim Coplien and Neil Harrison [4].

Lack of trust

Lack of trust is always a bad thing, no matter what is the methodology (agile or traditional) used. In fact, if trust is missing, honest communication becomes very difficult and so is keeping control of the project.

There are different types of lack of trust: between the customer and the team; between the customer and the project manager; between the project manager and the team; and between team members. The symptoms are, usually, not very difficult to spot, for example, when the customer (or the project manager) doesn't trust the team, often insists in giving technical hints and tips; or, when the project manager and the customer don't trust each other, often they insist in very detailed product specifications before starting any development, to be used, by any of them, as a weapon in case of problems. Finally, lack of trust inside the team, usually, manifests itself in the form of gossip at the coffee machine, or finger pointing (and, sometimes, scapegoating, usually against a former team member) when a bug shows up.

The fact that agile methodologies are mainly based on people and their interactions makes them even more sensitive to the lack of trust than traditional ones. For this reason, several "agile practices" such as collective code ownership, face to face communication, co-location, etc., are meant also to foster trust among all the stakeholders, but, unfortunately, they are not sufficient—I've been involved in at least one Extreme Programming project where all the above trust problems where present, even if we used all the practices suggested by the methodology.

There are no sure recipes for improving trust. However, besides some agile practices, there are some more things you can try.

First of all, everybody can contribute to creating a safe environment. This means that it should be safe for any stakeholder to express her concerns or criticism without having to fear humiliation or retaliation.

If you are a developer, a good starting point is to take responsibility for the code you write, and have the courage to be very clear and honest about what you can and cannot do without giving in to pressure. This last thing can be very difficult, especially at the beginning, but think about what happened the last time you didn't

deliver what you were “forced” to promise!

If you are a project manager, a good starting point is to trust your team, give them the resources they need, share the goals with them and allow them to take responsibility for achieving them.

If you are a customer, try to understand that there is a limit to what the team can do and, if you push them hard to do more without listening to their concerns, you will have to blame yourself for the bugs in the resulting product.

Measuring agility by the number of “agile practices” implemented

This is a very common mistake. First of all, most of the practices that are considered to be agile—e.g., configuration management, continuous integration, unit testing, collective code ownership, test driven development, etc.—are not specific to agile methodologies, but they are used in more traditional methodologies as well.

Furthermore, practices only make sense in a context, e.g., if the programmers in the team are not comfortable with pair programming, forcing them to do it could be a very big mistake.

Of course using appropriate practices is very important for the success of a project—for example, I couldn’t work without having configuration management in place—but the real difference between being, or not being agile is attitude—in an agile project there is a big emphasis on people, communication, trust, and collaboration. The tools and techniques are important only as long as they add value to the work, when they don’t add value any more they are discarded or adapted to the new situation.

If you want to introduce new practices in order to make development smoother and/or safer, again, it is better to look for ways to convince everybody involved (developers, customers, project manager, etc.) to buy into your idea.

Thinking that merely implementing some of the practices will improve quality

Unfortunately, this silver bullet view has been promoted also by several people in the agile community. In my opinion (and experience), none of the practices can automatically improve quality of the system, of the code, of the design or testing.

A case in point is Test Driven Development (TDD), which is writing the tests (usually the acceptance or unit ones) before writing any code.

Nowadays, it is often sold as the new silver bullet that will solve all of your code quality problems almost overnight.

This is a technique that, if used properly, can give excellent results in term of code quality, and productivity. However, it is just a tool, and, like any other tool, it can be misused: I worked in a project where TDD was used extensively from the beginning, and yet the code-base was of a very poor quality in terms of bugs and maintainability. The same goes for pair programming and all the other practices.

The bottom line is, good practices are welcome, but you have to use your judgement, experience, and a grain of salt before (and during) the adoption of any of them.

Focusing too much on the process and not enough on the product

This is typical of a team using a specific process for the first time. To a certain extent it is normal: after all, when you are learning something new you need to focus on it to see if what you are doing is right or wrong.

However, when the team starts to think along the lines of “the code is not good enough, maybe we are not doing enough <put your preferred practice here>” too often, it could be a sign of something else going wrong.

Of course, good teams think about the process, and change it to fit better their current situation, but they spend only a fraction of their time doing that. In my experience, when too much time is spent on the process, it is a sign that the team is looking for a scapegoat for their shortcomings: blaming the process is easy and safe, since nobody is going to be held responsible for failure.

Conclusion

Implementing an agile methodology can give many advantages in terms of product quality, customer satisfaction, and team satisfaction as well. However, it is not an easy job: customers may fight against it because they have to be more involved and take more responsibility for the outcome; Project Managers need to learn how not to be control freaks and delegate more authority to developers; and developers have to accept more responsibility and be more accountable for what they do.

For these reasons, using a checklist based approach is not going to make the team more or less agile. Even more importantly, don’t expect any practice or technique to magically improve the quality of your code-base—they are just tools that, if used wisely, may help; if not, at best won’t change anything, and at worst may have disastrous consequences (especially if their use is mandated from above).

The real change happens when all the people involved are given a stake in the product and in the process as well, and they are trusted to do a good job. This is the essence of agility.

However, it is quite simple to shoot yourself in the foot by inadvertently making any of the mistakes described in this article, but can be very difficult to spot them—especially when we are the ones to blame—but, if you keep an open mind, make it safe for others to give you honest feedback, and use it to correct your mistakes, then you are likely to have a very positive impact on the implementation of your agile methodology.

References

[1] Agile Manifesto web site http://www.agilemanifesto.org
[2] Agile Alliance web site http://www.agilealliance.org
[3] Rising, L., Manns, M., L., *Fearless Change: patterns for introducing new ideas*, Addison Wesley, 2004
[4] Coplien, J., O., Harrison, N., B., *Organizational Patterns of Agile Software Development*, Prentice Hall, 2004