AGILITY BEYOND THE DEVELOPMENT TEAM

EOIN WOODS
CTO

INTRODUCTION

Agility is a quality that many organisations want to achieve, many service providers claim that they are "agile" and even offer to "transform" their clients into "agile organisations".

Many of these exercises have disappointing outcomes, which while resulting in some positive change, do not fundamentally change the organisation as a whole.

In this article we will consider what we mean by agility and explore why, even when software development teams use agile practices, the attempts at agile transformation may not deliver the intended outcomes and what we can do about this.

WHAT IS AGILITY?

Agility is a widely used term and has a range of meanings for different people. Philippe Kruchten captured it nicely when he suggested that agility means

> *“ the ability of an organization to react to changes in its environment faster than the rate of these changes [1]*

The key point about this definition is that agility is an organisational characteristic, not a set of processes or something limited to software development teams.
To decide if we are achieving results by “being agile” rather than just following some processes and “doing agile” we need look at the results that the organisation is getting rather than whether the approach used by our development team (like Scrum [2]) is considered to be “agile”.

WHY THE DEVELOPMENT TEAM ISN'T ENOUGH

When most organisations start to think about improving their agility, often in the software development organisation, this is usually due to software developers who want to work more effectively and be more responsive for their business users.

This is certainly a great place to start, as software development agility is fairly well understood and there are lots of success stories and resources to guide teams on this journey, but let's see why it isn't enough.

Development teams usually begin to reshape themselves to work in a more agile way by understanding one of the popular approaches (Scrum and XP [3] being common) and reorganising the way they work as the chosen approach suggests. This usually involves:

- organising projects as timeboxed "sprints";
- having a direct link to an empowered "product owner" who can make decisions and resolve queries for the team;
- using a constantly reprioritised "backlog" of work, rather than an up-front plan, allowing the product owner to change direction as business needs change;
- implementing the system as useful, deliverable increments (known as "stories") to allow them to be used without waiting for everything else to be done;
- using a high degree of automated testing, to ensure that the software is working all of the time;
- refactoring code constantly to maintain quality and avoid technical debt;
- introducing "retrospectives" that allow the team to constantly revaluate and improve their process.

The results of well implemented agile development processes are often dramatically successful.

Rather than requiring heroic effort to produce occasional, unpredictable releases that are nearly always out-of-date with respect to user requirements, the team now produces regular, small, high quality updates and allows the end-users to change their requirements every sprint.

This is a terrific improvement on the situation that normally exists before the adoption of agile development.

After some initial pilot projects are completed though, difficulties often start to emerge and things seem to slow down and start frustrating people.

Some of the common problems are:

Infrastructure may be another reason why moving change to production is difficult. Unless they've been part of the agile transformation, the infrastructure teams are probably set up to focus on reliability and deliver change to quite lengthy SLAs – usually via a change request process. This can mean that even if the software could be released, it can't be until the infrastructure is ready (for example, when its database storage has been allocated).

Moving to production slows everything up drastically. The development team may be producing new reliable software every two weeks, but many organisations are set up to deliver to production 3 or 4 times per year. So the release management processes just aren't available to release software every two weeks – just running user acceptance testing often takes longer than this. The result is frustration as users can see the software is ready but may need to wait months before being able to use it.

Business units finding it difficult to deal with a constant flow of changes in their products and services and can become quite defensive when they realise that they are the new bottleneck in the process, not the development group!

Operations group processes may also not be optimised for rapid change and they may find a move to agile working very disruptive, unless they've been involved and able to consider the impact on how they work.

Marketing groups can find it difficult to adjust to marketing the overall value in a constantly changing product rather than a more traditional approach, which often uses the excitement of big new releases to create interest.

Business support functions are often forgotten during agile transformation, but they too can have a major impact on the success of an agile transformation.

Business functions like HR, Finance and Programme Management may inadvertently cause problems with agile working because they may expect everyone to work to classical 12 month planning cycles with little option for flexibility and change during the year – just what agile working is trying to avoid!

The **existing application estate** may also be a barrier to agility for two reasons.
Firstly, the existing applications may well be struggling under a load of technical and functional debt built up over many years and so may be difficult to change.

These applications may also be a major distraction to the development teams; if they're spending half of their time responding to L3 support requests for the older applications, they'll find it difficult to concentrate on agile transformation.

So are these problems inevitable?

Do we just have to accept that an agile transformation will be limited to the development teams and have a limited impact on the organisation as a whole?

Or is there a way to avoid these problems?

In fact, perhaps counterintuitively, rather than limiting the scope of the agile transformation, the way to make it successful is actually to extend its scope beyond the development team and help all of the organisation improve its agility.

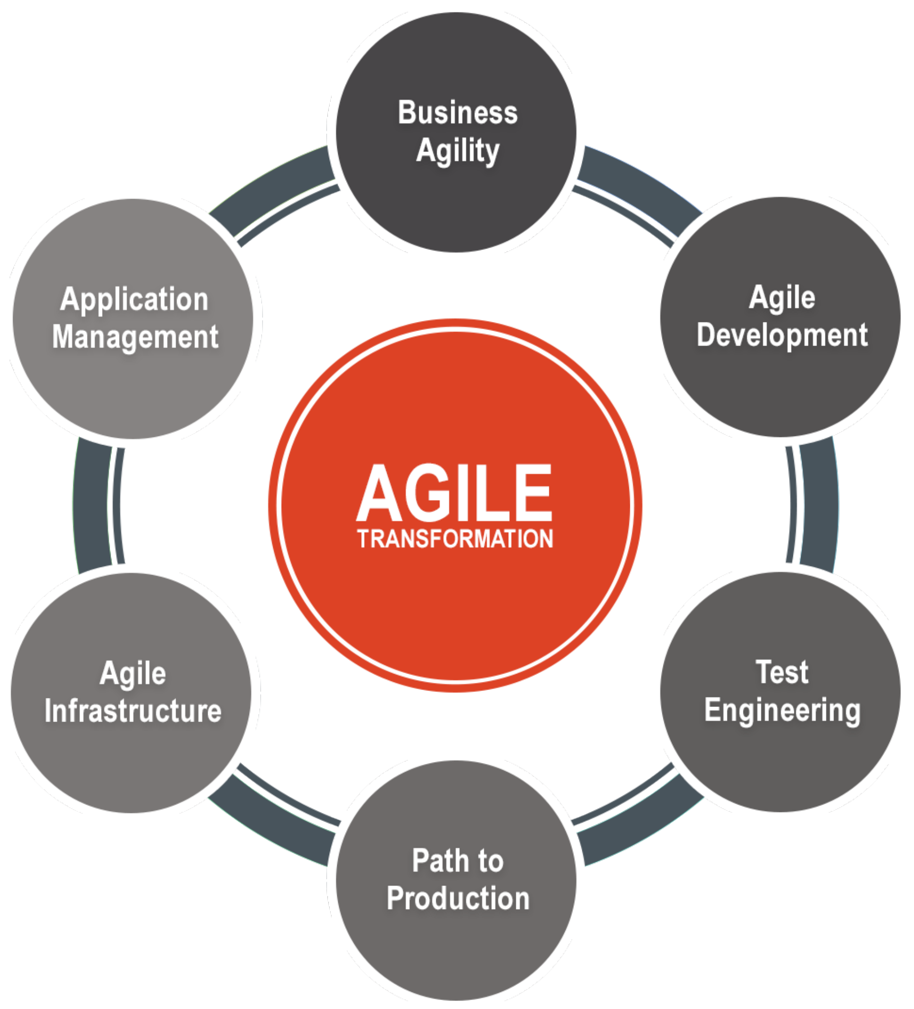

Figure 1

Dimensions of Agile Transformation

BEYOND THE DEVELOPMENT TEAM

Our experience suggests that there are six different aspects of an organisation that need to evolve in order for agile software development to have a real impact on the organisation.
We illustrate these areas in Figure 1.

BUSINESS AGILITY

As already mentioned, the wider business organisation outside IT needs to be able to respond to an agile IT organisation to support and capitalise on its new way of working.
There are two distinct aspects to this, the **lines-of-business** and the **business support functions**.

Firstly, the main business units, who are the customers of the IT group, need to understand how to work with an agile IT group and the opportunities and responsibilities that this implies. For example, an agile development organisation will not be very effective if the client business unit cannot provide informed, enthusiastic, knowledgeable and empowered product owners.

In order to capitalise on the results of an agile transformation in the development group, a business unit also needs to know how they will harness this new responsiveness and use the flow of small features that this implies to test ideas and move quickly in their marketplace.

Support functions within the business also need to be part of the transformation process, and provide agile workers with the support they need, otherwise they can end up being unwitting obstacles to its success.

“

Inflexible annual budget cycles, unsuitable working spaces and rigid HR processes that assume detailed annual plans are all common features of organisations that are likely to hinder agility.

AGILE DEVELOPMENT

The process of software development teams adopting agile methods is fairly well understood. There are a number of agile methods that are widely used and a very large number of development teams have been through this process.

However, as we've already discussed, this process doesn't always have the impact that is hoped for and there are a couple of other areas that moving to agile development involves. In particular, rather than "doing agile" development organisations need to learn to **"be agile"**.

Becoming agile involves changing the culture of the organisation and the way that people think and work, rather than just following a new process. Otherwise you can end up using the approach dubbed "Water-Scrum-Fall" where the process looks like Scrum but little has really changed.

To be successful with agile delivery, development organisations need to ensure that their core engineering practices are suitable for faster paced delivery. Techniques like Scrum deliberately don't talk much about this – they focus on the management process and assume you have the engineering practices right. However, agile development relies on software that works all of the time, can be released frequently and can be changed easily, requiring teams to master techniques like Test Driven Development, Continuous Integration, version management, quality measurement and refactoring.

And finally, for an agile transformation to be effective, a development organisation needs to develop multi-skilled teams who can work effectively with other parts of the organisation. This includes being able to work with business people in terms they can understand, as well as working with their testing and operational colleagues to create a reliable and efficient "path-to-production" for their software to minimise the delay from implementation to operation.

This requires the team to master techniques from different disciplines such as stakeholder management or scenario modelling from business analysis through to testing and DevOps type techniques like automated acceptance testing, automated non-functional testing and continuous delivery, with the automation and sophistication that it implies.

So achieving basic development team agility may be a well understood problem, but achieving effective development organisation agility is a rather more involved undertaking. spanning process, culture and engineering practices.

The modern test team integrates with the development and operational teams, as well as being responsible for some of its own testing.

TEST ENGINEERING

While we expect all developers to be competent automation testers and to be testing as a core part of their work, there's always the need for further testing, particularly when continuous delivery is being used.

Historically, testing has been seen as an activity that blocks the release of software according to rigid criteria, and is something performed by a siloed testing team, far away from the developers (intellectually if not geographically). None of this sounds very agile and, sure enough, testing also needs to respond to the challenges of agility.

The modern test team integrates with the development and operational teams, as well as being responsible for some of its own testing between the two.

Testers are experts at understanding the risks inherent in a piece of software and use their abilities with a wide range of testing techniques, including automation, non-functional testing, mobile device testing, exploratory testing and test management, to make this visible throughout the path-to-production.

PATH TO PRODUCTION (CD AND DEVOPS)

A common problem for teams undertaking an agile transformation is how to get their code from their continuous integration pipeline to production. Moving code through test environments and then eventually to production is a time consuming, labour-intensive and bureaucratic process; a huge delay in realising the value of the development team's work. The development team may feel they have done a good job, but until it gets to production, their work is adding no value to the organisation.

We need to achieve a smooth, reliable and efficient path-to-production for the code coming out of the development process. One way to achieve this is to aim for Continuous Delivery and to adopt DevOps practices for the most difficult step in the journey – the transition from development and test (uncontrolled) environments to production (the controlled environment).
The core idea of continuous delivery is to reorient everyone involved in delivering software, from the idea to its production operation, into a single virtual team with the same goal – getting the software running reliably in production.

This is a totally new way of thinking to many groups who are used to operating in their own siloes, using formalised handoffs with other teams.

Continuous Delivery needs a new way of working where everyone cooperates as a single project team. It does not mean removing control, it means cooperating towards a common goal. The trickiest part of this transition is usually between the development (and test) organisations and the operations organisation because they have such different priorities.

Development want to make lots of small changes and migrate them quickly to production. Operations are concerned with the stability and efficiency of the operational environment, and this usually drives them towards highly structured processes and minimising change. It is little wonder that this can be a fractious relationship! DevOps practices align key elements of how development teams and operational teams work, identifying shared processes, practices and tools so that operations people are deeply involved in the end of the development cycle and development people are deeply involved in the transition to production.
Modern shared tooling like Ansible [4] and SaltStack [5] and packaging technologies like Docker [6], help provide tools and approaches that both groups are happy to use and so foster collaboration on other aspects of the problem too like practices and processes to move code safely and efficiently from development to production.

AGILE INFRASTRUCTURE

In many cases, the software can be released without changes to the underlying infrastructure, but this is not always the case and where infrastructure change is needed, we find that an application centric implementation of continuous delivery has probably just moved the problem down the line, not removed it.
If infrastructure reconfiguration or change is required and this process is lengthy then once again we will lose the real agility that we are looking for.

Production environments need to be carefully controlled because poorly managed change will result in them becoming unmanageable and unreliable; quite the opposite of what we need to rapidly delivery reliable change. So we're certainly not trying to remove controls on the production environment, rather we need to find a way to increase the rate of change, while improving the reliability, efficiency and speed of the process.
Ironically, this may well involve removing some human access to production that is there to begin with, because a major part of making infrastructure environments agile is automation and standardisation.

When we standardise approaches and services and automate everything, we actually increase the rate of change due to the reduction in manual intervention required.

The tools and approaches to achieve agile infrastructure exist today.

Virtualisation of infrastructure, APIs for infrastructure management and automation tools for infrastructure configuration and management can all be combined with change-oriented agile processes to make the infrastructure environment as agile as the other parts of the organisation.

1

Gartner, somewhat simplistically, refer to a more flexible approach to infrastructure management as "mode 2" management [7] although we would challenge whether it really needs to reduce the emphasis on accuracy and safety as Gartner suggest.

APPLICATION MANAGEMENT

Like it or not, nearly all organisations have applications, often critically important to them, that were developed well before agile practice were common.

Even organisations that prioritise DevOps, often stop thinking about agility at the point when change is delivered to production, but this can easily end up ignoring a major barrier to agility – the existing application estate.
These applications are often difficult to change but encapsulate critical data and processes that other, newer systems rely on.

Application Management is the discipline concerned with supporting, maintaining and evolving applications after their release, usually after the bulk of the development of the system has been completed. This includes application support, incident management, and sustaining engineering (for fixes and extensions to the application).

Application Management can help an organisation to be more agile in two main ways.
Firstly, a proactive approach to managing and improving an application can significantly improve its ability to change, even if it will never be as agile as new applications developed with rapid change in mind. Secondly, explicitly moving applications into an application management phase of their lifecycle helps an organisation focus its change teams on the applications that it really wants to change quickly.

This helpsv clarify priorities and strategy and allows the agile transformation to focus on the most important parts of the application estate first.

CONCLUSION

...the prize is huge – the ability to be the fastest mover in the market and so achieve lasting success.

In this article we have discussed why an organisation embarking on an agile transformation needs to look well beyond techniques like Scrum and the software development teams if they are to achieve real agility.

Without considering organisation-wide change across business functions, software development teams, testing teams, operational teams, and even the way the application estate is managed, then an agile transformation is likely to be a disappointing experience that fails to deliver real business value, even if the development organisation manages to become more responsive to some of their stakeholders.

Such a transformation is much more challenging and involved than just adopting some agile practices in the software development teams, but the prize is huge – the ability to be the fastest mover in the market and so achieve lasting success.

REFERENCES

1. P. Kruchten, “Contextualizing Agile Software Development,” Journal of Software: Evolution and Process, vol. 25, no. 4, pp. pp.351-361, 2013.

2. K. Schwaber and M. Beedle, Agile Software Development with Scrum, Pearson, 2001.

3. K. Beck and C. Andres, Extreme Programming Explained: Embrace Change, Addison Wesley, 2004.

4. Red Hat Inc, “Ansible Documentation,” [Online]. Available: http://docs.ansible.com. [Accessed 08 03 2016].

5. SaltStack Inc, “SaltStack Documentation,” [Online]. Available: https://docs.saltstack.com/en/latest. [Accessed 08 03 2016].

6. Docker Inc, “Docker Documentation,” [Online]. Available: https://docs.docker.com/. [Accessed 08 03 2016].

7. Gartner Group, “Gartner IT Glossary > Bimodal IT,” [Online]. Available: http://www.gartner.com/it-glossary/bimodal. [Accessed 10 03 2016].