Ten tips for making your Agile adoption successful

By Allan Kelly, Director & Consultant

I am not the first to write a "10 habits of highly effective Agile adoption" type article and I'm sure I won't be the last. Actually, I put off writing such a list largely because there were so many such lists floating around.

Ken Schwaber has been reported as saying that only 30% of teams who attempt Scrum will be successful. On his blog he says he doesn't remember this and instead suggests only 30% will become "excellent development organizations."

What I find interesting about this quote is that it aligns with many other change management studies. Change experts like Harvard Professor John Kotter regularly say 70% of major change efforts fail.

However you look at it the prognosis isn't optimistic. Then one day as I was finishing a course delivery someone asked me: "What can we do to ensure that we are in the 30% who make it?" – at that moment I knew I had write this list.

There isn't anything magical about 10 items it could have been 7 or 12, all three numbers have a resonance but as I put the items together it seemed that these 10 were more significant than anything else I could think of, and taking anyone away made the list less effective.

So, with apologies for another top-10 list, here you are.

1) Use a physical board

> "I put the shotgun in an Adidas bag and padded it out with four pairs of tennis socks, not my style at all, but that was what I was aiming for: If they think you're crude, go technical; if they think you're technical, go crude. I'm a very technical boy. So I decided to get as crude as possible."
> William Gibson, Johnny Mnemonic

I have become convinced that the single biggest difference between teams which successfully adopt Agile working and those who try, fail, or end up stuck is the use of an actual physical board.

I know some teams find this difficult, I know some teams are distributed, I know there is technology out there to do this for you but I stand by my point. If you can make it physical, in a place where many, if not all, can see, then you are more likely to succeed.

It is hard to explain the logic involved in why I recommend this, you have to try it and witness it. I feel there is some kind of magic that happens when the very abstract, theoretical, world of software meets physical cards and board.

Visualizing the work is only the start: learn to read the board and act on what it is telling you.

2) Start collecting and using statistics

Velocity, burn-down, bugs identified, bugs logged, etc. etc. Metrics have a bad name in software development - rightly in many cases. But that only means that have been badly chosen, collected and/or used, it doesn't mean they aren't useful. At the very least measure your velocity and create a burn-down chart or cumulative flow diagram of the work.

The beauty of Agile working is that it is quite easy to measure a few simple metrics. Once measurements get complex people don't understand them so it becomes more difficult to learn from them. Like the boards, the metrics are useful in their own right but are far more useful as a learning instrument.

3) Engage a coach/consultant:

At the risk of being accused of trying to make work for myself I should say you can adopt Agile all by yourself. You can read the books, you can experiment, you can go on courses. But doing it without help makes the whole process slower and increases the risk that you won't make it to the 30%. Plus, adoption will take longer – and that brings more risk.

Personally, I find it difficult to know just how an Agile Coach differs from an Agile Consultant. What ever you call the role you want someone who can:

- Observe, examine, query and challenge your thinking on what you are doing
- Provide advice on which practices and process to adopt, and how to best adopt them
- Offer examples of what they have seen work, and not work, elsewhere, and how other team tackle similar issues

You may need to work with multiple advisors since few will be able to cover all process, practice, technology, product and strategy bases. On very large team it might be worth having full-time consultants although the model I have had most success with is light-touch coaching in tandem with a pull-change model (below).

I don't believe such an advisor needs to be full time but I do believe it should be ongoing. I practice, and have written before about, light-touch Agile coaching, in this model I return to companies at intervals, perhaps monthly, perhaps more frequently, sometimes less frequently and continue the discussion.

4) Action over talking

Action speaks louder than words, until you start trying to do Agile you can't foresee all the issues and questions which will arise. The longer you spend talking about doing it, and not actually doing it, the more it anticipation will build up, the more more it will look like jus another management fad.

By all means talk about it, plan a bit but there is no real substitute for just getting stuck in and doing it. Learn from what your doing and refine. Use some measurements to understand what is happening. Don't waste your time looking for evidence, make your own.

In particular do not spend your time agonizing over whether to do XP or Scrum, or Lean or FDD, or DSDM or Kanban. They are all pretty much of a muchness and you will end you up crafting your own hybrid anyway.

5) Give everyone training and start group wide discussions

Teams don't get to be Agile by management deeming "thou shalt be Agile" - although plenty have tried the approach. Reading books works for some people but most books go unread, or the words go in one eye and out the other.

Invest in taking the time to explain to people what Agile is – or at least what Agile means to your organization. Today most techies have heard the word "Agile" but what they have heard, which bits they remember, and whether the result was good or bad differs. Teams need to start with a shared understanding.

But don't stop there, make time for people to talk about what Agile is to them, what they like, don't like, will do, won't do. Agile is a team sport and unless the team have a shared understanding they will be playing different games. This discussion should start in the training sessions as the team forms their own, shared, understating.

6) Enthuse, Pull, don't Push:

Anyone who has worked around companies for a few years will have seen management pushing the latest change initiative: BPR, ISO 9000, Sig Sigma, CRM, etc. etc. Someone dreams up these ideas and then a change machine sets about pushing them out.

We live in a post-modern, post-BRP, post-layoffs, post-recession, post-everything world. Employees aren't children they've heard what happens. Too often before management initiated change, especially that involving consultants, has involved redundancies. If you want to cut staff then cut them then move onto Agile.

Apply a lean principle: Pull, don't push.

The moment someone uses the words "change management" you are in the world of push change so forbid the words "change management."

If you are in management this means you need to engineer a pincer movement: you want enthusiasm for change coming from the bottom up to meet your support coming top down. Introducing Agile top-down alone is,

in my opinion, quite likely to kill it - employees are rationally skeptical of top-down management change.

Seek to enthuse individuals and teams, have them ask for Agile. Rather than impose change from the top down managers need to build, kindle people's curiosity, get people asking questions and for help, create bottom-up change initiative and support it.

Do everything build the fire without extinguishing it, and when they ask give them the help and support they need: sign-off book expenses, provide budget for speaker, trainers or coaches, say Yes to time for conferences. Give and give generously.

Plus, involve yourself. You need to learn too – even if you know it all you need to be there when the shared understanding is built. And you need to change too, learn to change your own behaviors to match.

7) Be clear on Why you are going Agile

What ever level you are, engineer, tester, project manager, director, look beyond the Agile hype. What is the problem you want "Agile" to fix for you? Understand why you want change and what you expect from it.

Don't just "get Agile" because it is this month's fashion, get "Agile" to achieve something more important. Understand what individuals want from Agile, what they want to make their life better, and understand what the organization wants from Agile – innovation? Reliable schedules? Fewer bugs?

Ask you team, ask your manager, and ask "What do you think your manager wants?" If everyone stands to benefit from Agile then everyone will be willing to help with adoption. When people don't see benefits change will be hard.

As your adoption proceeds you use these aims to choose your tools, optimize your processes and measure your success.

8) Process and technical, Adopt technical side as well as process side

Don't think you can just change the process and it will all be all right. You need to address the technical side too, you need to improve quality, you need to support the engineers, testers and others who are at the code face doing the work.

I've come across big companies who view the technical side as somehow dirty: the attitude seems to be "thats technical" or " they get their hands dirty" or "we can ship it to [Low cost country of choice this week]". If this is your attitude you will fail.

Get your hands dirty, talk to engineers, adopt Test Driven Development, refactoring, shun big up front design architecture, learn to live with rough designs and evolving architecture. There are real feedback loops here.

9) Get requirements flow clear and clean

It isn't just about fixing the coding side, the requirements side needs to be addressed to. Specifically there needs to be a clear path from someone who represents requirements - typically called a Product Owner or Product Manager and frequently staffed with a Business Analyst - and the development team. Far more negotiation is going to happen over "what" then "when". Someone needs to represent - and have authority - over that side of things.

Too many companies understaff this role to begin with. Agile makes that understaffing acute. Bottlenecks in requirements will have a knock-on effect in development.

As a mental exercise as yourself: if Agile doubles developer productivity what happens next?

The answer is you need twice as much effort on requirements. O, at first you might just burn off a big backlog. But as you do so your marginal value delivered may well be declining.

10) Structural changes - Functional groups

Staff your teams to do the work for which they are responsible, end functional groups - i.e. database developers and UI developers in separate teams. This is just the first of more structural changes you will need to make. But if you fail at this you won't get to play again.

Teams which have to call out to other groups for specialist skills or authorizations will block. Other groups have other priorities. Little impediments build up, each one slowing the team down, sapping morale and making your adoption more difficult.

That's it

There you go, each of those items could be an article in its own right, maybe one day they will be. That's enough to make a difference.

If there was an eleventh is would be: let go of the past, things change, Agile isn't purely additive. If you don't stop doing some of your current things you will never see the full benefit. But that can wait until you've got some success under your belt.

Originally published on InfoQ, June 2012

http://www.infoq.com/articles/ten-things-agile-adoption;jsessionid=4BACD993B92B1C8AC1572EE696A7CCFD

About the Author

Allan Kelly is London based and works for Software Strategy where he provides training and consulting in Agile practices and bespoke development services. He specialises in working with software product companies, aligning company strategy with products and processes.

In addition to numerous journal articles and conference presentations he is the author of **Business Patterns for Software Developers** (2012) and **Changing Software Development: Learning to become Agile** (2008). He is also the originator of Retrospective Dialogue Sheets (www.dialoguesheets.com). On Twitter he is @allankellynet and his blog is at http://blog.allankelly.net.

About the Software Strategy

Software Strategy Ltd. is UK based software development consultancy specialising in Agile development processes and practices. Software strategy provides training, consulting and coaching in software development and offers bespoke software development.

Clients range from small web development agencies proving custom solutions to the IT departments of large corporations. Most clients are UK based while others are in Europe and the USA. Clients include Virgin Atlantic Airways, Thompson-Reuters, Iris Legal, Research Instruments, Budget Group Insurance, Fugro and many more.

More about Software Strategy at www.softwarestrategy.co.uk, e-mail contact@softwarestrategy.co.uk or telephone +44 (0) 20 3286 4292.