

The Four Change Initiatives of Agile Adoption

© N. Van Schooenderwoert, 2007, all rights reserved

Now that the world has heard of Agile, they think – incorrectly – that the pieces of Agile they like best can be cherry-picked and used in isolation. Unless it is combined with Lean Thinking, Agile software development can achieve only a fraction of its potential. Agile software teams are not sustainable for very long if they are islands in a sea of waterfall projects. The presence of Agile teams creates a new and incompatible dynamic within a waterfall company. Agile adoption programmes conducted without a thorough understanding of this dynamic will continue to have a very high mortality rate, especially in larger organisations. This paper describes four change processes that must occur simultaneously for an agile adoption programme to succeed.

High Mortality of Agile Adoption

Why do we see such a wide variation in agile adoption programmes across companies? Why is it that we so often see agile pilot projects able to deliver better quality software in significantly less time, and yet the companies involved do not replicate this success across the enterprise? Every year the Agile conference has many experience reports showing high quality projects delivered in 30% to 50% of the time they would have needed using previous methods. From the agile coach community (no verifiable statistics, unfortunately) I know that a very large percent of companies fail in their attempt to take agile the rest of the way to being used enterprise-wide.

Often when a company has piloted a few successful Agile teams, managers decide that they can mostly go on as before and just “cherry-pick” some of the Agile practices and tools. The “mostly go on as before” part means conducting product development as a *push* system, rather than as a lean pull system. So rather than bother with having a Product Owner (the person who pulls value from the development team on behalf of the business), they let analysts decide via committees what features to have the agile team build. Testing is difficult to arrange for realistic scenarios so rather than invest the effort, they settle for superficial testing that is too labour-intensive to be sustained. These are just a couple typical compromises that pave the way for failure.

There are actually four change initiatives that must be managed successfully if a business is to sustain and spread the success of agile pilot teams throughout the enterprise. Even more critical is that they must occur simultaneously. They are:

- Teams must be able to produce bug-free software sustainably
- Teams must consist of empowered, engaged people
- Work flow to the agile teams has to be controlled via a ‘pull’ system
- Lean portfolio management must be used to control work flow for the organisation

When these change initiatives are not all occurring as they should, characteristic problems arise. Let's look in on an agile team a year after they completed a transition to agile.

Agile Erosion

The Comet team is one of several agile teams that were set up in the course of an agile adoption programme at an insurance company. Their story here is a composite of events that I and other agile coaches have seen. They had external help for a total of 8 months: Scrum training, and additional technical training in the use of agile test frameworks and help in cleaning up their build process. Their consultant Scrum Master was on site most of the time for the first 3 months and then present every other week until disengagement when they felt confident to continue on their own.

Comet Team from Agile Adoption to One Year On

When the consultants completed their work, the 8 members of Comet team were running 3-week iterations. Their velocity had initially jumped around but was stable and gradually increasing. They were holding retrospectives at the end of every iteration and checking a few days into the next iteration to make sure they were really following the new resolutions they made in the retrospective.

Their Product Owner was 50% allocated to that role and was meeting with stakeholders a week prior to the end of each iteration to finalise stories for the backlog. The team's ongoing mission was to maintain and enhance risk analysis software for use by the underwriters. The work they turned out was the best quality the company had seen. The team knew that there were still bugs in the legacy code but at least they were not adding many new ones. As they improved their tests they were cleaning out more and more old bugs.

A year later the consultant Scrum Master saw Tony, the Comet team's lead developer, at an industry conference and asked him how things were going. Tony said he wished they had never gotten into Agile at all! They were no longer doing real iterations, just moving the work along in a more or less continuous stream. They had dropped doing the retrospectives after Joanne, who facilitated them so well, moved on to other work. They continued them awhile but they were just making a quick list of what worked, didn't work, and what they'd change. Another problem with the retrospectives was that the things they most needed to change required cooperation from another department that was always overworked. So there seemed no point in making the same useless resolutions over and over. No one from that group ever came to the retrospective although they were invited.

Dissension Within Team

The biggest letdown in Tony's mind was that Agile had opened the door to turning software development into a sweatshop. That's the word he used – "sweatshop". Before the conversion the developers had their own cubicles, and in the enthusiasm of early Agile they'd given them up in favour of a team room. But they were happy before with the team room – why the problem now? They were arguing a lot now, and being together all the time in a team room only increased the tension.

Stressors

They were arguing a lot because of disagreements over how much time to devote to cleaning up quick fixes they'd had to put into the code versus getting new functionality in place that their customers needed. At least that's when the arguing began. Now there were many running arguments on all sorts of things. The quick fixes got rushed into place when new features activated a couple latent bugs in the code that their agile test framework didn't yet cover. Those bugs got a set of policies written with incorrect risk assessments, forcing Legal to sort out a remedy. That brought criticism on the project manager who'd been fine with team autonomy so far. But now he was being criticized by his manager for not having been firm enough with the team.

Repercussions on Team and Other Roles

The Product Owner had cut way back on the time he devoted to his role because the team was delivering so predictably for months before the problem with the bugs and the Legal department. He felt that at some point Agile should just be able to run on autopilot, and there were many other demands on his time. He was feeling caught-out when Legal had to come to the rescue. At first he defended the team, understanding their explanation about the test framework. But he became dismayed by their incessant arguing and no longer wanted to be in the Product Owner role.

Both the Product Owner and the Project Manager were starting to wish for the good old days when you just had the analysts write a specification and hand it over to a virtual team – one where the Project Manager rotated people in and out of a dispersed team as needed. It was getting easy to forget the problems associated with that, when these new problems seemed so intractable.

Team velocity was decreasing, and that caused management to put more pressure on them. With each iteration, the team responded by promising more and delivering less. In an effort to work faster, they no longer helped each other out, working solo on the tasks they claimed for themselves. Everyone missed the teamwork they used to have but didn't know how to get it back again.

Tony was using the conference to renew his contacts elsewhere. He had decided to leave the company.

Diagnosis

It is clear that in this agile conversion this team learned how to operate using agile practices – to an extent - but managers did not learn how to monitor this new Agile system and use the leverage points that could have kept the situation from spiraling downward. Pleased by the agile turnaround, they expected it to just remain on course of its own accord.

Agile systems don't automatically stay on track. No system does; they all need some degree of monitoring and adjustment. By "Agile system" I am referring to the core team as well as their stakeholders and the other groups that support them – sometimes referred to as the project community. After an agile adoption initiative is completed, managers need to have learned how they fit into this new system and how to carry their new roles.

Let's look at a simple analogy. If enforcement of traffic laws were suspended, it wouldn't be long before there would be chaos on the roads. People who want to follow the laws would have to

respond to those who don't and the whole thing would spiral downward. Likewise in an organization it is for managers to understand the laws – the key leverage points – and use them correctly to keep the agile system running with minimal ad hoc intervention.

Let's take it one step further. Suppose there has been a breakdown – as we've seen with the Comet team – what is management to do? An accident, flooding, or downed trees can create a big traffic jam on a road. Regardless of the cause, the individual motorists are powerless to move their vehicles once they are caught in the gridlock. No amount of pressuring or punishing *them* makes any sense. The first thing the police do is *not* to pull cars out of the jam, but to halt the inflow of more cars. They block off the routes into the gridlock, thus preventing it from growing. Next, they free cars out the other side by clearing the blockage.

What should be done in the case of the Comet team? The set of problems can be viewed in terms of the four change initiatives.

Problems With Change Initiative #1: Sustainable Bug-Free Code

Agile teams must be capable of delivering bug-free software. That doesn't mean zero bugs every time, but it does mean that they should be negligible. Troubleshooting and fixing bugs should not take up a noticeable amount of an agile team's time. Agile technical practices are designed to achieve this goal, and keep the code base in a fit condition for further work. Agile technical practices shift the mindset from:

Specify → code → integrate → then do an infinite test & fix loop

to

Test via specification → code → integrate continuously so code is always releasable

Agile teams must be able to estimate their work and – in the absence of blockages – be able to deliver on their estimates every time. Managers are partners with the teams. They fulfill their part by promptly moving blockages, and by providing the team with resources to fully test their code during development.

Let's look at Comet's bug issue. Activation of bugs in the legacy code via new features added to the code base was the initiation point for the unraveling. "Technical Debt" is a name for the flaws that lie hidden in software. All software has latent bugs, or technical debt. But code written the agile way with strong unit tests has far fewer such bugs – typically one or two orders of magnitude fewer. In Comet's case, technical debt was getting removed incrementally. This is normal. Removing it all in one go would mean having the focus only on that for some time without building new features.

Without their strong unit tests fully constructed, the possibility of a critical bug getting through does exist. Even if the unit tests were fully constructed, it is always possible to overlook something in the design of the unit tests. Human error cannot be eliminated completely by Agile or any other methodology. With that said, agile teams routinely achieve bug rates on the order of 0.2 defects per function point, or roughly 2 bugs per month for a team of 5 people.¹

Management in this case needed to own the risk of bugs. The risk can never be taken to 0% but it can be lowered to any level that one cares to pay for. This is always a cost-benefit decision that is ultimately up to management, not the team. They decide how good a test environment to provide. They decide how much testing effort to pay for. Even if a perfect testing environment is provided, there is no limit to the number of agile unit tests that can be written for a given code base. You can make the tests as fine-grained as you need. But they will require effort (cost) to maintain.

Agile technical practices give the team new skills to take the risk far closer to zero than they could have before. Still, it is for management to own the fact that accidents can happen, and they must be prepared to untangle the resulting traffic jam if an accident does occur.

Particularly in software development, insufficient early investment takes a very heavy toll. The problem is that it's hard to say definitively just how much specification detail is enough, or how much requirements analysis is enough, etc. Agile teams solve this problem by following a simple rule: In every iteration deliver tested, working code that customers can directly evaluate. That means deliver whole usable features, not mere technical components.

Every agile team needs to be able to produce bug-free code. To the extent that you have bugs, your project is out of control.

Problems With Change Initiative #2: Empowerment

Partnership between team and stakeholders is fundamental. If one side can never say 'no' to the other then it is not a partnership. This is what opens up the entire domain knowledge of a whole team and places it fully in service to the organisation. Only autonomous teams can take full ownership of a goal, relieving managers of tedious control tasks. 'Servant Leadership' refers to a philosophy of indirect control, of empowering those you seek to lead.ⁱⁱ

An agile coach or scrum master is someone who uses servant leadership to help the group achieve its aims. Servant leadership is facilitative rather than directive. This doesn't mean "indecisive". It is a technique that fosters a high degree of ownership over goals. It is superior to a command-and-control style because it allows decisions to be decentralized - made by those most in touch with the situation.ⁱⁱⁱ

When a manager decides to become a servant leader, he or she can get put into a bind because the boss is still doing command and control, and expects that to be transmitted down the line. Not only that, but servant leadership requires a lot of patience and trust. It can feel very much like a loss of control because it involves dropping the old methods of control in favour of new ones. This is very difficult to do and should not be underestimated. Spending time with a team can help to accelerate the shift, assuming of course that the team is deserving of trust.

Empowerment of the agile teams is not optional; it is crucial for the high degree of focus and dedication that is necessary to produce bug-free code sustainably. Mere process rules that a company may institute cannot make people engage to the degree necessary for producing such high quality software. Their creativity and energy are needed – not just their obedience to rules! No externally-imposed discipline is any match for self-discipline. This is the key to Agile (and to

Lean) which is completely missed by so many would-be adopters.^{iv} It is also key to the genuine professionalism that distinguishes real agile teams from pretenders.

Let's look at how problems with empowerment surfaced for the Comet team. There are many problems stemming from this, and in my coaching experience that is usual – empowerment is widely viewed as a “nicety” that can be skipped, or even worse as a step toward anarchy.

Erosion of “Soft” Skills as First Danger Signal

The first danger sign was the disappearance of retrospectives. They are not the only possible mechanism for it, but group learning and team unity of purpose *must* be renewed periodically. This is not an optional thing. It is a very high-leverage activity that solves a great many thorny problems while they are still tiny. The line manager(s) of those on the team should have noted that Joanne was carrying the load for retrospective facilitation, and should have recognized the importance of that skill by helping others get trained to do it well also.

A related mechanism is good facilitation of team consensus decision-making. This should have prevented the team's arguments from getting out of control. The first argument was over how much effort to spend cleaning up the quick fixes in the code when customers were clamouring for new features. A wise technical manager knows that new features built on top of quick fixes just creates a bigger cleanup job to be done later. It also makes bugs much easier to create. If this wasn't being grasped by the Product Owner, then it was a point where an IT manager could've reinforced the team's message. Too often good facilitation is dismissed as a “soft skill” not as important as the technical skills, and something we can do without in a pinch.

Missing Cooperation Between Departments

Failure to act on issues raised in retrospectives is another problem. The team had repeatedly raised issues requiring the cooperation of another department, but it was not forthcoming. By letting this issue sit, management sent a message to the team that said their issues are not important. This contributed to the team's feeling that retrospectives were pointless.

In an agile adoption programme, friction between the agile teams and the surrounding organisation *will* occur. The organisation evolved to support waterfall teams and siloed job functions. It will have to change if the agile teams are to survive. Management's chief job in an agile adoption programme is to watch for the friction points and then *adapt the organisation to the agile teams*.

Misalignment of Accountability and Control – ‘Blame’ culture

The organization was trying to hold the project manager responsible for the team's actions. The team has to be responsible for its actions. There was nothing the project manager could've done to make the team avoid those bugs. When things go wrong there is often an impulse to place blame. The need to deflect blame is what lies behind much of the overly-detailed planning in waterfall projects.

In the case of the bugs that occurred, they were unforeseeable since they were latent in the existing code. Probably the only way to have prevented them was if the company had invested in more thorough testing designed to catch a scenario of that type. But it is very common (and possibly

justified) for companies to decide not to invest in doing that. In that case, the team were operating within a system that had set this trap for them. Once the bugs had repercussions in the Legal department, the organisation's blame reflexes kicked in. Too bad they didn't have the benefit of a good retrospective to identify these dynamics and find a constructive way to address them!

Failure to Sustain Agile Feedback Mechanisms

Another key leverage point that management neglected to use is the extremely vital feedback loop between team and stakeholders. A strong Product Owner actively engaged with both the team and the stakeholders is necessary for agile teams. Like well-run retrospectives, this prevents a host of problems that are very difficult to address in firefighting mode. Once a team loses the trust of their stakeholders everything is an uphill battle. A truly agile company would recognize that although projects vary in their demands on Product Owners, it is always wise to invest in keeping that feedback loop healthy.

The Product Owner role is new with Agile, and it calls for a person knowledgeable about the product vision, who has relationships already established with stakeholders. Such people are already busy, and it is difficult to convince their superiors that they must be permitted time to carry out the role properly. Product Owners usually end up doing too little communication with the team or with the stakeholders. Often they skimp on the initial training and do not fully grasp that their role is to *actively* manage the flow of *value* from team to stakeholders.

Product Owners need to be empowered to perform their role. They must be given goal-setting authority by the project's sponsor. They need to respond to the team's questions with little chance of being overruled. They must be able to arbitrate stakeholders' needs also without likelihood of being overruled. They need this authority so they can provide the team with information that won't be changing within the iteration, and so protect the team from having to waste time on rework. Product Owners often get put into an impossible bind when the organisation does not support their role.

Summary of Empowerment issues

In summary, Comet's management needed to do these things to prevent the problems stemming from insufficient empowerment:

- Recognise the importance of facilitation skills and grow those skills
- Act on issues arising from Retrospectives that are outside team's control
- Support the need of Product Owners for time to perform duties
- Support the Project Manager when he/she gets inappropriate pressure to direct the team
- Insist on continuing with well-run retrospectives or "lessons learned" sessions

Without the right empowerment, the other goal of sustainable bug-free code cannot even be achieved. These two change initiatives are very much bound up together. A properly empowered agile project community is *the means to* generating sustainable bug-free code.

Problems With Change Initiative #3: Team's Work Stream

An agile team 'pulls' work from the Product Backlog into a time-boxed iteration. After a couple iterations they know how much they can do in a given period; they have established a velocity. A common problem for managers new to agile is that they want the team to do more than the amount their velocity indicates. So they pressure the team to give artificially low estimates, or to work long hours, and so on.^v

Velocity as an emergent property

An essential idea of a pull system is it recognizes that the team cannot control everything that has a bearing on their velocity. The team is part of a bigger, interconnected system. That system can produce bug-free code at a certain rate, full stop. That's the velocity. It might be stated as so many story points per week. If faster output is needed then the interconnections between the agile core team and the rest of the system need to be understood and improved to generate a higher velocity.

Velocity is an *emergent property* of the system. It cannot be directly commanded. If you get 50 story points this week by working 12 hour days, when the usual was 40 points, you are not working sustainably. You're simply robbing future production to get a peak now. Perhaps a tool to do data profiling would result in higher velocity. Tools are an example of an interconnection between the core team and the organisational system. So are skills. If the team is given training to help them write better agile unit tests, that could improve velocity in a sustainable way. So could having a good team workspace. Each idea has to pass the sustainability test – if you could do it indefinitely then it's a valid way to influence velocity.

A very common mistake is to try to drive velocity directly, through pressure.

Diagnosis of Comet's Work Stream Problem

Comet team's velocity eventually started decreasing. When more coding is done on top of quick fixes, the code base quickly becomes brittle, and very hard to work with. Changes produce new bugs and they take time to fix, hence a slower speed. An increasing bug rate tempts people to do even more quick fixes leading to a downward spiral.

In frustration over the developing problems, Comet's managers reacted by pressuring the team to get more work done. Direct pressure cannot solve this – it intensifies the problems. The team responded by overly-optimistic "commitments", and by refusing to devote any time to helping each other with tasks. Undoubtedly, that further slowed their progress.

The team could not, in a short period of time, clean all bugs out of the legacy code. It was not within their control to do that; it was part of the landscape they had to cope with. They also could not quickly cover all the legacy code with agile unit tests. It had been agreed that this would be done incrementally along with creating new code.

In this instance, managers needed to recognise that the velocity was slowing because the group had decided to go with using quick fixes, and they were not taking the time to clean these out of the code. The question managers needed to ask is “why are we using an unsustainable technique to keep code production going at this rate?” How was the team’s consensus mechanism being used? Was the team simply bowing to inappropriate pressure to keep velocity up? That’s the surest way into trouble. Or did the team really believe the quick fixes would be harmless? If so then they should have seen that wasn’t the case and reversed course. Retrospectives provide a good way for managers to gain early insight into issues like these before they are compounded.

Lessons Learned for Keeping Team’s Work Stream Healthy

Agile teams should receive their priorities through the Product Owner, and no one else. Sometimes standards groups, regulatory representatives, or others expect to direct team activities. They are stakeholders in the team’s outputs and should work through the Product Owner. When a team has too many bosses they are set up to fail. Note that the Product Owner does not *direct* the work of the teams. They are self-directing. The Product Owner sets the targets and is an information resource for the team.

Pressure on a team to make optimistic estimates is often non-verbal, and it may come from many sources. Teams *want* to deliver, and when they err it is always on the optimistic side. It doesn’t take very much pressure to move them onto shaky ground. Learning to push back in an appropriate way is one of the last skills agile teams acquire, so new agile teams are especially vulnerable to this mistake.

Wise managers will watch for times when a team commits to do much more work than they usually accomplish, and they will ensure that the team know they’ll be supported if they need to push back on demands made of them. A commitment that cannot be fulfilled is of no help to the organisation but can become a magnet for blame. Managers must understand that they have a responsibility to regulate the flow of work to agile teams. It is a simple, effective way to help teams keep their credibility.

The worst thing an agile team can do is erode trust with their Product Owner. Every iteration should build up that trust.

Agile teams that form spontaneously “under the radar” are usually undermined because management has not acknowledged any obligation to regulate their workload. They are flooded with too much to do, no time for learning new tools or for maintaining a unit test infrastructure, and they can only hold the line so long.

If teams have mastered bug-free code and are empowered, they can still be destroyed by the failure of management to regulate the flow of work to them, i.e. allow them to pull the work in at their sustainable pace.

Problems With Change Initiative #4: Organisation's Work Stream

One of the worst things that can happen to a good Agile team is to be assigned to a weakly supported project. That will mean blockages don't get moved. People and other resources get taken away when needed by other projects. The team repeatedly fail to deliver on their commitments and they lose the trust of their Product Owner.

The previous three change initiatives discussed are covered by the popular agile methodologies, but this issue of governing the work stream at the organisational level takes us over to Lean territory. Unless a lean discipline is used to decide what work an organisation undertakes, it runs the risk of spreading its energy too thinly.

The organisation must commit itself to doing only the work it has capacity for. That means saying no to some things. Someone has to take responsibility to say the company's resources will not be spread too thin. Weakly supported projects will thrash and waste the company's resources. A company that regularly completes 40 to 50 projects a year should not have 500 active projects in their portfolio!

Projects that have a direct business need are easiest to do as agile because there is a strong *pull* by a customer. Necessary infrastructure projects are harder to gain sponsorship for. There is less will to allocate a dedicated team, and pay for the tools they'll need. There is no easy answer here, but if the project is truly needed then it is the responsibility of the business to understand what it will accomplish and why. It is the responsibility of the business to own that goal and to be a customer for it.

In the example of the Comet team, they were one of several agile teams. There isn't enough context to tell for sure whether theirs was a weakly supported project. Probably not, as their software was for use by the insurance underwriters to determine risks. This points out that it's not enough to just look at individual agile teams – you need to understand how groups of agile teams function within an organisation.

Agile Habitat

One agile pilot project can be sustained by almost any company. It will place more or less demands on various other departments and infrastructure, and can be coped with. When the number of agile teams increases (the number will of course vary with company size) then pressure on certain resources reaches a point where something has to give. Either the agile teams will be reined in and forced to make more and more compromises in their roles and practices, or else management will recognize that they have to make the organisation into a better habitat for agile teams.

An agile team that has nothing wrong with it may still fail. It's the same problem as when healthy animals are released into a habitat that is being decimated. They don't have a real chance to survive. At any point in time an organisation can only sustain some number of agile teams. This is an emergent property of the organisation, just as velocity is an emergent property of an agile team.

Managers need to understand the dynamics of this and figure out what changes to make in order to sustain agile teams. They will get the time to do this by allowing the agile teams to be self-organising, and by exercising frequent, light control. In other words, attend the retrospectives, remove blockages daily and address every issue while it's still small. They will also have to

regularly kill off the waterfall projects in the portfolio that cannot deliver or that are vying with agile projects for resources. This is what it means to exercise lean portfolio management.

If teams have mastered bug-free code, are empowered, and their work stream is properly controlled, they can *still* be destroyed by the failure of management to regulate the flow of work through the organisation.

This is why Lean and Agile need to be used together for creating software-intensive products and services.

Conclusion

The Comet team got into quite a difficult situation even though they were in good shape when the external coaching ended. Once problems accumulate to this degree, it is very difficult to sort them out – too much so for an organization still getting used to Agile. The key is to prevent these problems by understanding the control points in an agile system.

Too often it is easy to dismiss those control points as they involve so-called “soft” skills like good facilitation, or because people are under too much time pressure to attend a retrospective. Agile adoption must be led by managers who thoroughly understand these dynamics. Otherwise the normal frictions that arise will completely erode the agile adoption programme.

Agile is not just new software development techniques, or new project management techniques. It is the tip of an iceberg that addresses both these areas and much more. Agile points the way to applying lean thinking to software-intensive product development. Lean businesses are radically different from traditional businesses in almost every respect. This is why it can be so difficult to move an organisation fully to lean-agile methods. It’s also why big rewards still await those who get there first.

ⁱ “Embedded Agile Project by the Numbers With Newbies”, paper presented at Agile 2006 conference by N. Van Schooenderwoert. Available at <http://www.leanagilepartners.com/publications.html>. The author found many other agile teams who had similar successes but had not produced detailed metrics.

ⁱⁱ “Servant Leadership” by Robert K. Greenleaf, Paulist Press 1977.

ⁱⁱⁱ “Primal Leadership” by Daniel Goleman, Harvard Business School Press, 2002. Command-and-control leadership is appropriate for some situations described in this book.

^{iv} Quotation: "Only after American carmakers had exhausted every other explanation for Toyota's success - an undervalued yen, a docile workforce, Japanese culture, superior automation - were they finally able to admit that Toyota's real advantage was its ability to harness the intellect of 'ordinary' employees." Source is "Management Innovation" by Gary Hamel, Harvard Business Review, February, 2006.

^v There are techniques to achieve higher velocity, such as using multiple agile teams for the project, changing composition of the team, etc. This is beyond the scope of this paper.