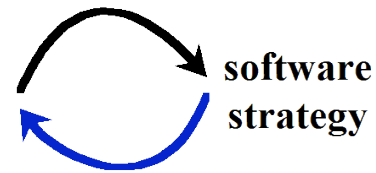


Agile Demystified (v.3)

A brief explanation of Agile Software Development for managers



By Allan Kelly, consultant and author of
Changing Software Development: Learning to Be Agile (2008).

Contact allan@allankelly.net or visit www.allankelly.net

Objective



Within the software application development community *Agile* has created a buzz. The term has been around for about nine years now and the ideas behind it slightly longer. But, many IT managers and directors are still confused about what Agile actually is. This paper will attempt to clear up some of this confusion.

Contents

A brief explanation of Agile Software Development for managers.....	1
Objective	1
Business value	2
Applicability	5
Multitude of methods and terms.....	7
Test of Agile	8
How Agile works	9
Agile myths.....	10
What is not Agile.....	11
Failures	11
Where to begin	13
About the author.....	14
Glossary of Agile terms.....	15

Business value

The debate over whether Agile actually delivers benefits is largely over. While there are some projects which do not benefit from the Agile approach most do. Gartner group said in 2006:

"It's a fact that agile efforts differ substantially from Waterfall projects. It's also a fact that agile methods provide substantial benefits for appropriate business processes. Separating these facts from the fiction surrounding agile development is crucial for an application development (AD) organization to achieve those benefits."


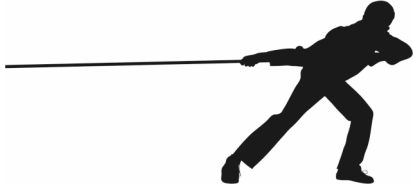
Bodies like the Project Management Institute are moving to reconcile their approach with Agile and the Carnegie Mellon Software Engineering Institute issued a report late last year showing how CMMI and Agile are compatible (Glazer et al., 2008).

Compared to earlier – so called "Waterfall" or "Traditional" – development techniques Agile offers a number of business advantages:

- Enhanced responsiveness to customer needs because Agile methods are designed to accommodate changing requirements.
- Increased return on investment because projects deliver working software earlier in the development cycle.
- Reduced risk because frequent deliveries of software both amortize and exposes risks and forces them to be tackled.
- Improved quality because high quality actually underpins Agile processes.
- Better project governance because projects progress is more transparent. The incremental delivery model used allows progress to be observed directly. Should the need arise a project may be terminated early and still deliver valuable software.
- Greater productivity: Agile methods foster high performing teams. A report on Agile adoption at Yahoo (Benefield, 2008) claimed teams improved productivity between 35% and 400%.



Table 1 - Summary of Agile benefits

 <p>Push to Agile</p>	<p>Pull to Agile</p> 
<p>Traditional methods:</p> <ul style="list-style-type: none"> • Requirements freeze creates unresponsive projects • Record of late and over budget delivery • Poor quality record • Delivery schedules do not keep pace with demands of modern business • Expensive and administratively heavy processes 	<p>Agile methods:</p> <ul style="list-style-type: none"> • Responsive to business need • Deliver working software earlier and continue to do so • Incorporate customer feedback • Higher return on investment, improved cash flow • Reduce risk • High quality software • Improved project governance • Lightweight nature makes it easier to start and stop project

Traditional software projects tended to deliver software at the end of the project in a single release or *big bang*. This gave rise to the cash flow profile shown in Figure 1. In such a project costs are incurred during the lifetime of the project but benefits are only realized at the end of the project. Even this is something of an idealized view because such projects are usually followed by a “bug fixing” phase. Applications are released to users, who report bugs which need fixing.

In contrast Figure 2 shows a similar Agile project, in such projects deliveries start much earlier. Although the initial releases lack functionality they contain enough to be usable, so benefits start flowing far earlier.

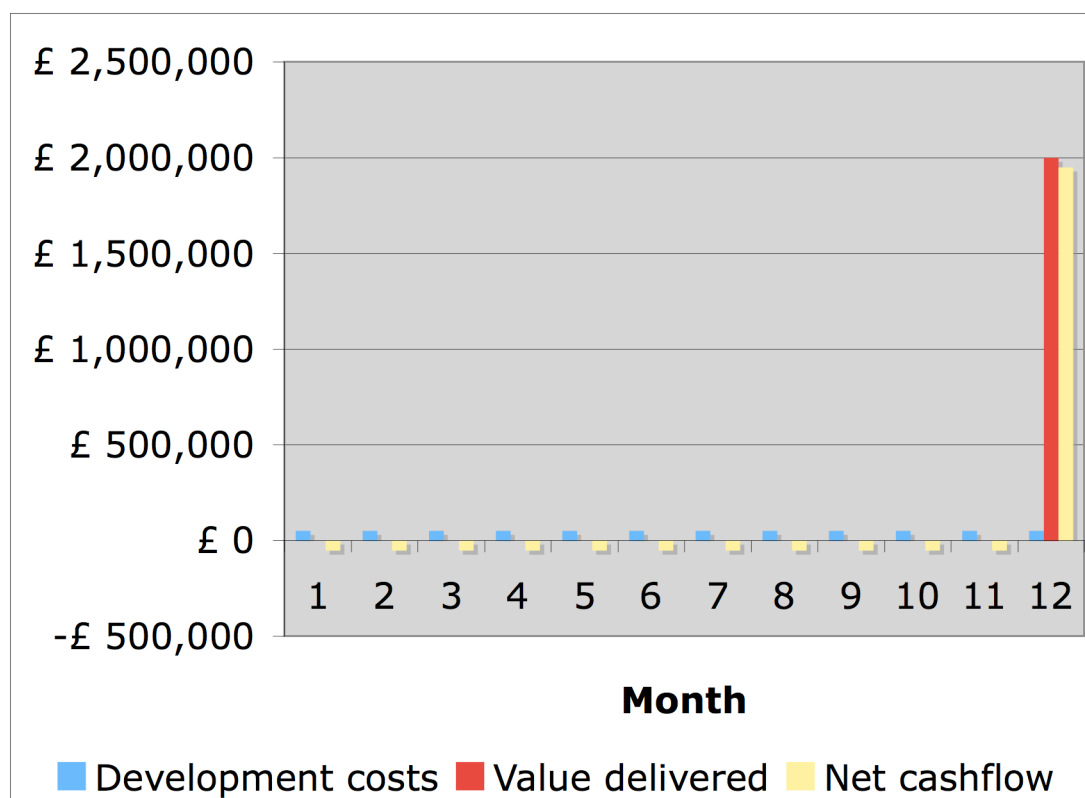


Figure 1 - Cash flow on a traditional Waterfall project

New software is released regularly increasing the benefits month by month until the same level of benefit is reached as the previous chart. In this case a number of additional benefits flow. With the software in use there is more information on which to base decision on which features to priorities for implemented. This in turn creates a closer match between business need and delivered features.

Secondly, because users are actively using the software issues become apparent far earlier and can be addressed before the end of the project. Team leaders can then decide whether new features or addressing issues will deliver greater business value.

Finally, IT governance is improved because managers have more strategic control over projects. Should it become necessary to close a project before the scheduled end-date benefits will still accrue because something has actually been delivered. In a traditional project, closing the project early usually leaves only part implemented, buggy, software of little value to anyone. (However, some approaches to project governance assume a Waterfall approach and will need updating if the full benefits of Agile are to be realized.)

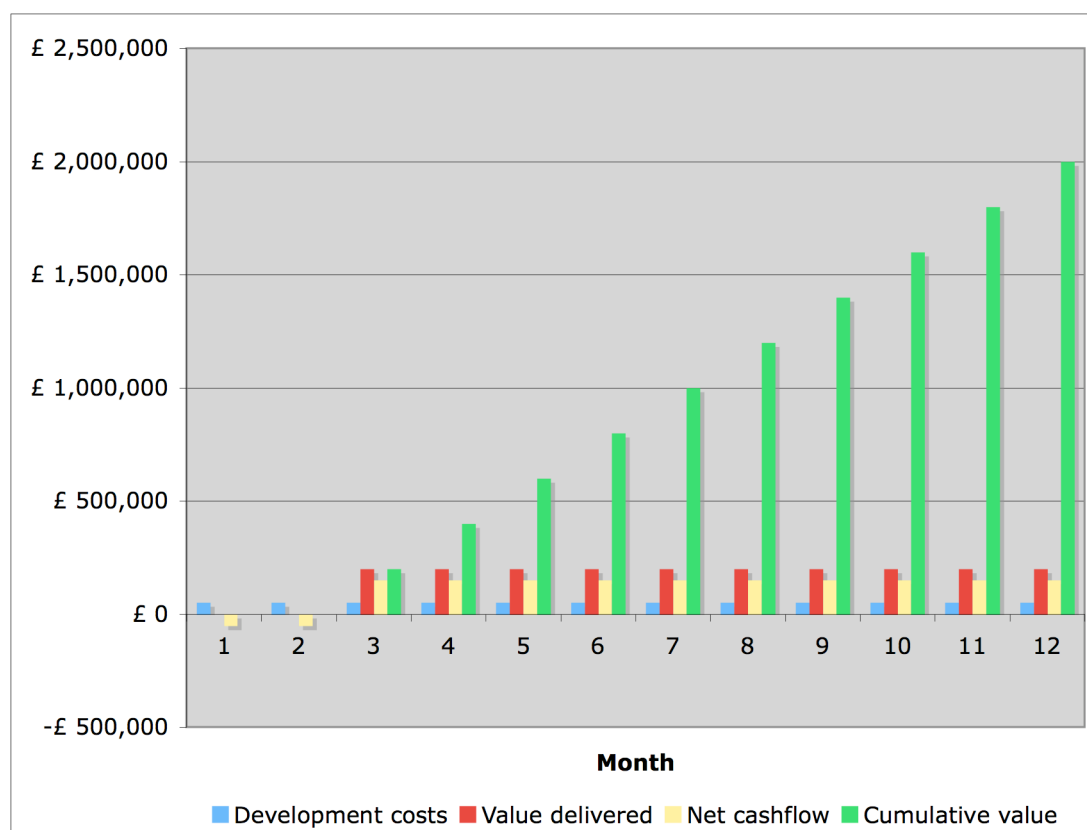


Figure 2 - Cash flow for an Agile project

Figure 1 and Figure 2 are generalized examples; both make similar assumptions about cost (£50,000 per month) and benefits (final benefit of £2m). With a discount rate of 5% per annum this yields a net present value of £1.318m for the Waterfall project shown in Figure 1 and a slightly higher NPV £1.354m for the Agile project in Figure 2.



The added value is shown more explicitly by an internal rate of return of calculation. Here the Waterfall Figure 1 has an IRR of 20% but, because benefits start flowing sooner, the Agile Figure 2 has an IRR of 100%.

These calculations make no allowance for improved productivity or higher quality that should also result. Thus the true value may be higher still.

Applicability

Agile methods are widely applicable. Most organizations producing software can adopt Agile methods. A better question to ask is: *when is traditional Waterfall development applicable?*

Waterfall development depends on fixing requirements early in the cycle. Once requirements are fixed then analysis proceeds. When analysis is complete design can proceed and complete before coding starts. Coding is usually the longest phase followed by testing. Thus delay in fixing the requirements delays all stages of the project while changes to the requirement has a significant ripple effect.

In contrast Agile methods expect requirements to change and emerge as the project proceeds. They accommodate this phases are overlapped by adopting a *just in time* approach to analysis and design.

(In fact, while many organization claim to follow the waterfall process-fixing requirements is incredibly difficult. Changes occur during the entire development period. One of the difficulties faced by teams has been reconciling the model with what actually occurs.)

At the time of writing Agile methods are most commonly used by media organizations, Web 2.0 companies and independent software vendors (ISVs). Agile methods have been shown to work in banks, telecoms companies and elsewhere.

The adoption of Agile by media and Web 2.0 companies reflects their post-millennial interest in software development. Prior to 2000 media organizations had little need to develop complex applications to support their core business. The development of such applications by media and Web 2.0 companies started to occur around the same time as Agile methods emerged. With no legacy development processes it was natural that these organizations adopted Agile.

The lack of a legacy code base and practices makes the adoption of Agile easy for these companies. For organization which already have established application development teams adopting Agile needs to be viewed as a change programme.

Major users of Agile methods include IBM, the BBC, Sky TV (the European equivalent of Fox in the New International portfolio), Nokia and Yahoo. Other organizations having pockets of Agile development include Google, Hewlett-Packard and Schlumberger. Many bank are experimenting with Agile methods and at least one large UK investment bank is moving towards widespread adoption.

Many of the practices found in Agile methods were to be found in the ISV community prior to the appearance of Agile, XP, Scrum or any of the other methods. Unlike companies which develop software to support their real business, ISVs live or die by their ability to create software products. Therefore there were already using many of the *best practices* which became Agile.

Still, there are many ISVs which could benefit from Agile methods and many more which could add to their already good practice with additional techniques.

Multitude of methods and terms

As with other branches of IT the Agile movement has a plethora of terms, methodologies and abbreviations: XP, Scrum, DSDM, Crystal, FDD, etc. (A longer list is contained in the glossary below).

Figure 3 shows how the methods and terms fit together. In each methodology there are some specific practices and routines, some are more prescriptive than others. In addition, each methodology brings its own philosophy, concepts and values.

The most prescriptive of all the Agile methodologies is perhaps the first edition of Extreme Programming, XP for short (Beck, 2000). Although Beck outlined some principles and values the original description was highly prescriptive. In the second edition (Beck and Andres, 2004) the prescriptive element was reduced with a greater role values and principles over practices.

Still XP is just one of several Agile methodologies. While Crystal Clear is less prescriptive than most it is still specific, as are Scrum and DSDM.

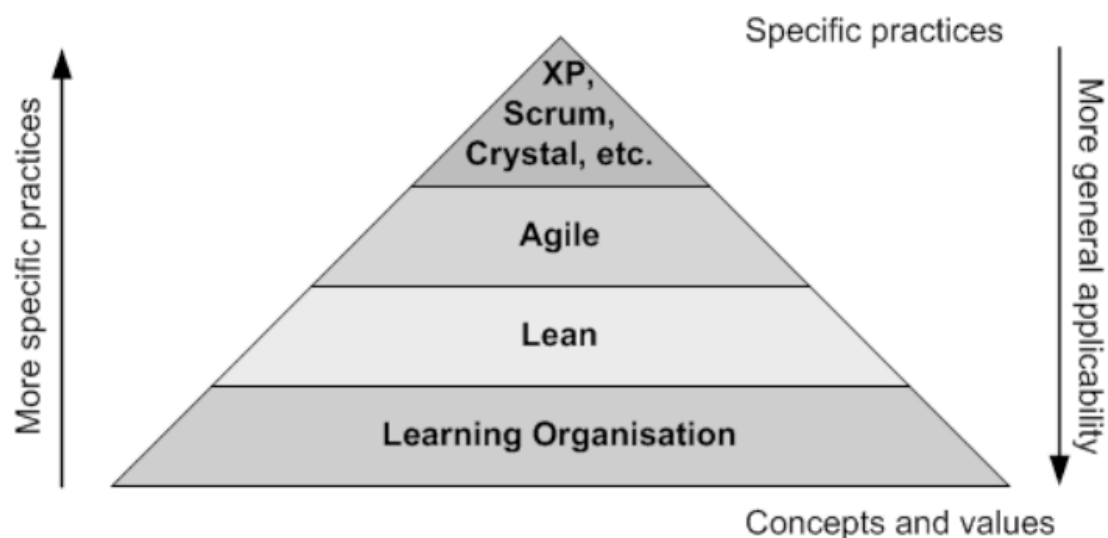


Figure 3 - How the Agile methods fit together

Agile as a whole is both an umbrella term to group all these methods – which were originally called lightweight methods – and a toolbox. While the Agile toolbox contains many prescriptive practices the toolbox user is left to select which tools to use. Consequently concepts and values are more important.

Lean, and specifically Lean Software Development, could be viewed as another Agile methodology similar to XP or Scrum. However, Lean is more concerned with how companies and teams improve and adapt. Although it has specific practices – like value stream mapping – these are concerned with process improvement. Lean is not so much a methodology of working as it is a method for improving working practices.

From this perspective Agile is an application of Lean thinking. Many ideas from Lean manufacturing, such as *just in time* production and a *quality is free* ethos underpin the Agile approach.

One of the originators of the term Lean, Professor Dan Jones, stated at the XP Day 2008 conference in London that he saw no difference between Agile and Lean. For most purposes it is reasonable to consider Agile software development to be the software industry's version of Lean product development.

The original description of Lean came from the motor manufacturing production line (Womack et al., 1991) which might make it seem unsuitable for the more abstract work of the software engineer. Elsewhere it has been shown how Toyota – and others – have applied Lean principles to product design and development (Kennedy, 2003, Cusumano and Nobeoka, 1998) while Mary and Tom Poppendieck have applied the ideas directly to software development (Poppendieck and Poppendieck, 2003, , 2007).

Finally, under pinning all of these techniques are the ideas of *Organizational Learning* and *the Learning Organization* (Senge, 1990, de Geus, 1997, Argyris and Schön, 1996, Kelly, 2008). While organizational learning can be a somewhat academic subject it explains how these techniques work and offers insights into how to manage the processes.

Test of Agile

Because there is no single source for Agile, and because there are so many variations on Agile itself it is increasingly difficult to know if a team is, or is not practicing Agile. One approach is to simply look at the practices described by Agile methods and examine whether a team is using them.

However this approach measures what is done rather than outcome so is less than satisfactory. Indeed it is likely that in a few Agile practices can be found in any development team.

Consultant Bas Voode invented a simple test for teams at Nokia to assess their adoption. This test asks two questions, each with several conditions:

1. Are you doing Iterative Development?
 - Iterations time-boxed to less than 4 weeks
 - Features tested and working at the end of each iteration
 - Iteration starts before specification is complete
2. Are you are doing Scrum?
 - You know who the Product Owner is

- There is a product backlog: Prioritized by business value, with estimates created by the team
- Team generates burn-down charts and knows velocity
- No project managers (or anyone else) disrupting the work of the team

This author proposes a more general test:

If a team can answer yes to the following questions it may be considered to be Agile:

- Is the team responsive to customer needs? Is it delivering business value?
- Is the team continually learning and improving? Specifically: the team should be changing the way it works as a result of its own learning over time.
- When the *change agent* – e.g. project leader, consultant - is removed does the team continue working Agile? Or does it fall back to the prior norms?

The ultimate test is not *Is the team Agile?* but *Is the team serving the business?* Too often IT becomes the block to organization agility and change.

How Agile works

All Agile methods, XP, Scrum, Crystal, etc. work by shrinking the development cycle and repeating it frequently. Each cycle is called an iteration or sprint and lasts between one and four weeks. New software may be released at the end of a single iteration or after several.

In effect they take bite-sized chunks off the problem rather than try to tackle it all. The result is a series of mini-projects in rapid succession. In order to do this the team need to reduce both the set-up time for a cycle and curtail the closing phase.

The set-up period is reduced by close customer involvement – where the customer may be an actual customer, a customer-proxy, a business analyst or a product manager – and rigorous prioritization.

In any one iteration the team is closely focused on a few high priority items. These will be completed in the iteration thereby allowing new prioritizes to be set for the next iteration.

Most software projects end with a test-fix-test cycle which is of indeterminable length. To avoid this, and thus shrink the closing phase, Agile methods adopt a *quality is free* approach and institute a number of techniques for boosting quality throughout the



development cycle. Techniques such as Test Driven Development (TDD), pair programming, automated acceptance tests and continuous integration help keep code quality far higher than on traditional project.

The net result is to remove the indeterminable test-fix-test cycle. When done right projects are always in a releasable state. This allows the mini-project to close on time with a deliverable.

Additional techniques are used to allow the project to cope with system architecture, maintainability and long range planning.

Agile myths

A number of myths have grown up around Agile. Some stem from the developer centric origins of Agile while others mistake the lack of traditional processes and artifacts for a lack of rigor.

One of the criticisms that has incorrectly been levied at Agile is that it is chaotic. In fact Agile is a high discipline process: it demands attention to technical quality, regular communication and planning.

Unfortunately, some teams that are chaotic excuse their behavior by claiming they are "Agile." Developers who refuse to show progress, demonstrate working code, write unit tests or listen to what the customer wants are not Agile. Such teams are taking advantage of the ignorance of others about Agile to excuse them from professionalism.

Another myth is that Agile is anti-documentation. Agile projects produce as much, or as little, documentation as is requested. However Agile teams do not produce documentation for the sake of documentation.

Others have questioned Agile applicability to safety critical systems. Again this is a myth. In some ways Agile is more suited to safety critical applications because of the continual emphasis on working code. In healthcare, pharmaceutical, embedded and elsewhere, there are Agile teams working on safety critical application.

Similarly it is untrue that Agile prohibits distributed teams. Like other methods Agile has a strong preference for co-located teams but dispersed Agile teams exist and successfully deliver.

Yet another myth is that Agile is only applicable when the developers are highly experienced. Project with highly experienced and skilled developers have an obvious advantage but Agile methods have been shown to work with average staff.

Similarly, it has been claimed that Agile is only suitable for new development, that it is not suitable for existing legacy applications. While it is true that legacy applications present their own challenges it

is not true that Agile cannot be used for such work. Indeed, the basic Agile approach has its roots in application maintenance.

What is not Agile

The success of Agile in recent years has resulted in a number of suppliers claiming their products, tools and services are Agile. This makes it more difficult to know what is Agile, and what is not.

- SOA (Service Oriented Architecture), MDA (Model Driven Architecture) and Virtualization are not themselves intrinsically Agile. An Agile project may use, or even deliver, SOA, MDA or Virtualization but the use of any (or all) of these techniques on a project does not make a project Agile by itself.
- Web 2.0, SaaS, (Software as a Service), AJAX (Asynchronous JavaScript and XML), REST, Mash-ups: Again, such technologies may be used by an Agile project, or they may not. Their presence does not make a project Agile or prevent it from begin Agile.
- Test First or Test Driven Development is a key Agile practice but not sufficient alone enough to make a project Agile.
- Pair Programming: Extreme Programming (XP) suggested that programmer work in pairs, a little like airplane pilots. This idea has some very vocal supporters but it has even more vocal opponents. Unfortunately the debate about XP or Agile in general, often gets bogged down in a discussion of *pairing*. If a team is willing to try pair programming great, try it, if not accept it and move on.

Failures

Agile is no a guarantee of project success. All IT projects, and especially application development, entail risk. If a project was risk free it is unlikely to provide significant benefits or competitive advantage. What Agile can de-risk projects and increases the return. Companies may take these benefits or choose to increase the risk in other areas.



There are however, a number of ways in which Agile projects repeatedly fail which are worth examining:

- **Wagile:** a team which continues to follow a basically Waterfall project but uses the language of Agile and adds a few of the artifacts or practices. For example, the team may present burndown charts together with Gantt charts.
- **ScrumBut** describes a team which claims to follow Scrum but misses various practices; for example "We do Scrum but we don't have a Product Owner" or "We do Scrum but the Project Manager

allocates tasks.” Such teams normally have a long list of “buts” and show little progress of removing them.

- **Hitting The Scrum Wall:** The most popular Agile method at the time of writing is Scrum which is a project management technique. Scrum is normally used with a number of other Agile techniques, typically User Stories and the technical practices from XP (TDD, refactoring, continuous integration, etc.).

Teams that adopt Scrum project management initially see an improvement in productivity and customer satisfaction. However without the technical practices quality is low and the team *hit the wall*. The quality gap makes it impossible to maintain the pace in the long run.

- **Fake Agile:** this occurs when a team declares itself Agile and blames everyone else for their failure to interact correctly with the group. Such a group typically stops writing documentation, listening to business analysts, product managers and other customers, dictates its own delivery schedule. Meanwhile the team do not improve quality, does not adopt test driven development or any other practice they dislike.
- **Potemkin Agile:** occurs when a team adopts and applies an Agile method well but does not deliver business value. This is a form of goal deferment where the team consider *adhering to the process* rather than delivering business value as the success criteria.
- **Customer (Business Analyst, Product Manager, Product Owner) overload:** on a well functioning Agile project the customer, or proxy customer, is called upon to do a lot. They need to decide requirements, set priorities, scout ahead of the project, align strategy, work with the developers, testers and managers, and may even have their own day job to do. In the earliest XP project (“C3”) the first business analyst came close to a nervous breakdown. Such overload is a sign that a project is functioning well but also a limitation.
- **Fall back:** management may bring in consultants and other experts help switch a team to Agile. When the consultants leave some teams return to their old ways of working. Advisers and consultants can be a great help when introducing Agile but they need to build capacity in the development team to continue learning and evolving when the consultants are gone.
- **Failure to go far enough:** To maximize the benefits of Agile Software Development the people, processes and organization that interface and work with the Agile team need to understand Agile and adjust their expectations and working techniques too. Agile is not a drop-in technology that can be swapped in to replace another failing method. Isolated Agile teams will find it difficult to be

completely Agile. When other groups adapt the benefits of Agile can spread beyond Software Development.

- **Exploding cards** happens when teams do not sufficiently understand the technology they are working with – either in the solution or problem domain. As a result small work packages turn out to be large tasks in their own right.
- **Hyper changing requirements:** With the exception of Kanban, most Agile methods, especially Scrum, hold the iteration goals fixed for a few of weeks. Most businesses should be able to hold to goals for such short periods of time.

If it proves impossible to hold requirements and goals fixed for even one week then something is wrong. In a few cases the business is genuinely changing extremely rapidly. When this is the case teams are better off using Kanban style management than a Scrum based approach.

More often hyper change in goals and requirements are a sign that something is wrong beyond the team. The organization itself may lack strategy and objectives, or the Product Owner may not be filling their role adequately.

- **Fragile not Agile:** some of the Agile techniques, like TDD, when poorly applied with a lack of understanding can show short term benefits but create long term problems.

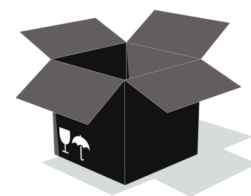
Few of these failure modes are unique to Agile; they are reoccurring failure modes for all IT software development projects. Neither is this a comprehensive list of the ways in which Agile, or any other application development, project can fail.

Where to begin

There is more to adopting Agile than simply declaring a team Agile. Neither managers nor developers can impose Agile by decree. Adoption is a learning process.

Ideally the adoption of Agile methods should be a pincer movement: Management should provide top-down support for adoption by way of training, consultants, and a wiliness to change themselves. Software development teams should launch a bottom-up initiative to change their own practices and methods of working. Both sides need to engage in shared learning.

Managers who wish to see their teams adopt Agile need to do more than just evangelize the techniques. They need to provide teams with the tools and resources they need to change. They also need to involve themselves closely with the change initiative by listening to developers.



It is not necessary to choose in advance which methodology to adopt. Each organization has its own needs, problems and demands. No *off the shelf* methodology will address the corporation's needs exactly, instead teams need to create their own methods from the available techniques to match their problems. This approach has the added benefit in that it will seed the creation of the learning culture needed for improvement in the longer term.

Organization can, and do, adopt Agile methods without external help however this is a slow and risky process. For faster adoption it is advisable to use the services of an Agile Coach to navigate the adoption process and guide the teams. Training in Agile methods and technical training – especially in TDD – is essential to embed a common understanding of the approach and skills required.

On the whole developers are keen to adopt new methods and try Agile. Management needs to work with this enthusiasm rather than impose top-down process change.

If you would like to know more about Agile Software Development, how your organization can benefit from becoming more Agile and how to migrate to Agile please contact the author, Allan Kelly on +44 773 310 7131 or allan@allankelly.net.

About the author



Allan is a regular conference speaker and contributor to publications on the subject of Agile development and improving software development. His first book, "Changing Software Development: Learning to be Agile" was published by John Wiley & Sons in 2008.

Allan holds BSc degree in Computing and an MBA in management. He has experience both as a development manager and as a software developer. He is a trained product manager and project manager – holding PRINCE2 Practitioner status.

Allan currently works as a consultant and trainer helping companies organize their software development activities and adopt Agile. He can be contacted at allan@allankelly.net and his personal website is <http://www.allankelly.net>.



Glossary of Agile terms

AD		Application Development
ASD		Agile Software Development
Automated Tests	Acceptance	Tests written by the Product Owner, perhaps with a Tester, which can be automatically run against software and systems. These form part of the program specification.
Blue-White-Red		An example Agile system by the author which combined elements of Scrum with XP (Kelly, 2007, Kelly, 2008).
Coach		Agile teams often include an Agile Coach. The Coach has a key role to play during the transition to Agile methods but is also responsible for helping the team reflect and improve their practices in the longer term.
Continuous Integration		The practice of integrating new source code as soon as it is complete and running system builds and unit tests many times a day.
Customer Customer, Owner)	(Onsite Product	XP mandates that each development team work closely with an Onsite Customer, Scrum fills the same role with a Product Owner. These roles are usually staffed either by an actual customer, a Business Analyst or a Product Manager.
Crystal		A family of methods from Alistair Cockburn.
Crystal Clear		
Crystal Orange		
Crystal Red		
DSDM		Dynamic Systems Development Method: a technique developed in the UK by the DSDM Consortium. This method has its roots in Government projects. Initially the use and documentation of this method was only available to DSDM consortium members, this has changed recently and DSDM Atern is freely available. As with Scrum some DSDM training leads to certification which is controlled by the consortium.
DSDM Atern		A new methodology from the DSDM consortium which is freely available.
EVO		Evolutionary project management from Tom Gilb. Sometimes called the "Grand Farther of Agile

		methods", Gilb and EVO have been around longer than other methods (Gilb, 2005).
		EVO enthusiasts claim it covers aspects of development not covered by other Agile methods and can be usefully combined with Scrum and XP.
FDD		Feature Driven Design a methodology from Jeff De Luca and Peter Coad.
Iteration		A short period of time during which work is performed. Iterations are "time boxed" in that they have a defined start and end, and all iterations are the same length. Work is then sized to fit the iteration time box.
Kanban		The newest Agile method: introduced by David Anderson about 2007 Kanban draws more directly on the ideas of Lean. Unusually for an Agile method the most advanced Kanban teams do not use time boxed iterations or give estimates. (The term "Kanban" has a specific meaning in Lean and its use to name a method causes a little confusion.)
Lean		Derived from the Toyota Production system as described in "The Machine that changed the world" by Womack, Jones and Roo.
Lean Development	Software	The application of Lean manufacturing and product development the software field. Most closely associated with Mary and Tom Poppendeick.
Organizational Learning		A branch of management theory concerned with understanding how organization learn and change, and how this can be used to inform operations and strategy. Most closely associated with writers like Peter Senge, Arie de Grus and Chris Argyis.
Product Owner		The team member responsible for determining what needs doing and prioritization. Role is usually filled by a Business Analyst or a Product manager, in corporate IT departments and ISVs respectively.
Refactoring		A technical practice used by Agile teams to improve the design of the software as they work on it.
Scrum		A methodology developed by Ken Schwaber and Jeff Sutherland. "Scrum" does not stand for

anything, it is a reference to game of Rugby.

The Scrum Alliance certifies Scrum training in the areas such as Scrum Master and Scrum Product Owner.

Scrum focuses more on project management while XP is more concerned with developer practices. This makes it natural to use elements of both together, as in Blue-White-Red (see above).

Scrum Master	Scrum defines a new role of Scrum Master which is designed to help the team over come obstacles and improve. In part the Scrum Master is an Agile Coach. While many organizations see the Scrum Master as a Project Manager this is not how the role is defined. The juxtaposition of Scrum Master as Project Manager can itself create tension.
Sprint	In Scrum: Iteration, or a collection of several iterations which make up a release.
TDD - Test Driven Design Also known as: Test First Development, Example Driven Design	Test Driven Design – originally part of XP, now a technique widely used in its own right.
XP	Extreme Programming – a methodology from Kent Beck (Beck, 2000) with Ward Cunningham (Cunningham, 1996) and Ron Jeffries. XP was initially the leading Agile methodology. This position has now been assumed by Scrum.

Graphics: Graphics taken from iStockPhoto, figures authors own work.

References

- ARGYRIS, C. & SCHÖN, D. A. (1996) *Organisational Learning II*, Addison-Wesley.
- BECK, K. (2000) *Extreme Programming Explained*, Addison-Wesley.
- BECK, K. & ANDRES, C. (2004) *Extreme Programming Explained: Embrace Change*, Addison-Wesley.
- BENEFIELD, G. (2008) Rolling Out Agile at a Large Enterprise. *Hawaii International Conference on Software Systems*. Big Island, Hawaii.
- CUNNINGHAM, W. (1996) EPISODES: A Pattern Language of Competitive Development. IN VLISSIDES, J., COPLIEN, J. & KERTH, N. L. (Eds.) *Pattern Languages of Program Design*. Addison-Wesley.
- CUSUMANO, M. A. & NOBEOKA, K. (1998) *Thinking Beyond Lean*, Free Press.
- DE GEUS, A. P. (1997) *The Living Company*, Nicholas Brealey Publishing.
- GILB, T. (2005) *Competitive Engineering*, Butterworth-Heinemann.
- GLAZER, H., DALTON, J., ANDERSON, D., KONRAD, M. & SHRUM, S. (2008) CMMI or Agile: Why Not Embrace Both! Hanscom AFB, MA, Software Engineering Institute
- KELLY, A. (2007) Blue White Red - an example agile process. *ACCU Overload*.
- KELLY, A. (2008) *Changing Software Development: Learning to Become Agile*, John Wiley & Sons.
- KENNEDY, M. N. (2003) *Product Development for the Lean Enterprise*, Richmond, VA,, Oaklea Press.
- POPPENDIECK, M. & POPPENDIECK, T. (2003) *Lean Software Development*, Addison-Wesley.
- POPPENDIECK, M. & POPPENDIECK, T. (2007) *Implementing lean software development : from concept to cash*, London, Addison-Wesley.
- SENGE, P. (1990) *The Fifth Discipline*, Random House Books.
- WOMACK, J. P., JONES, D. T. & ROOS, D. (1991) *The machine that changed the world*, New York, HaperCollins.