

Agile Demystified

A brief explanation of Agile Software Development for managers

Allan Kelly - www.allankelly.net

Within the software application development community *Agile* has created a buzz. The term has been around for about eight 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.

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.”

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 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.

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 cashflow profile shown in Figure 1. In such a project costs are incurred during the lifetime of the project but benefits are only realised at the end of the project. Even this is something of an idealised 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 Agile projects deliveries start much earlier. Although the initial releases lack full functionality they contain enough to be usable, so benefits start flowing far earlier.

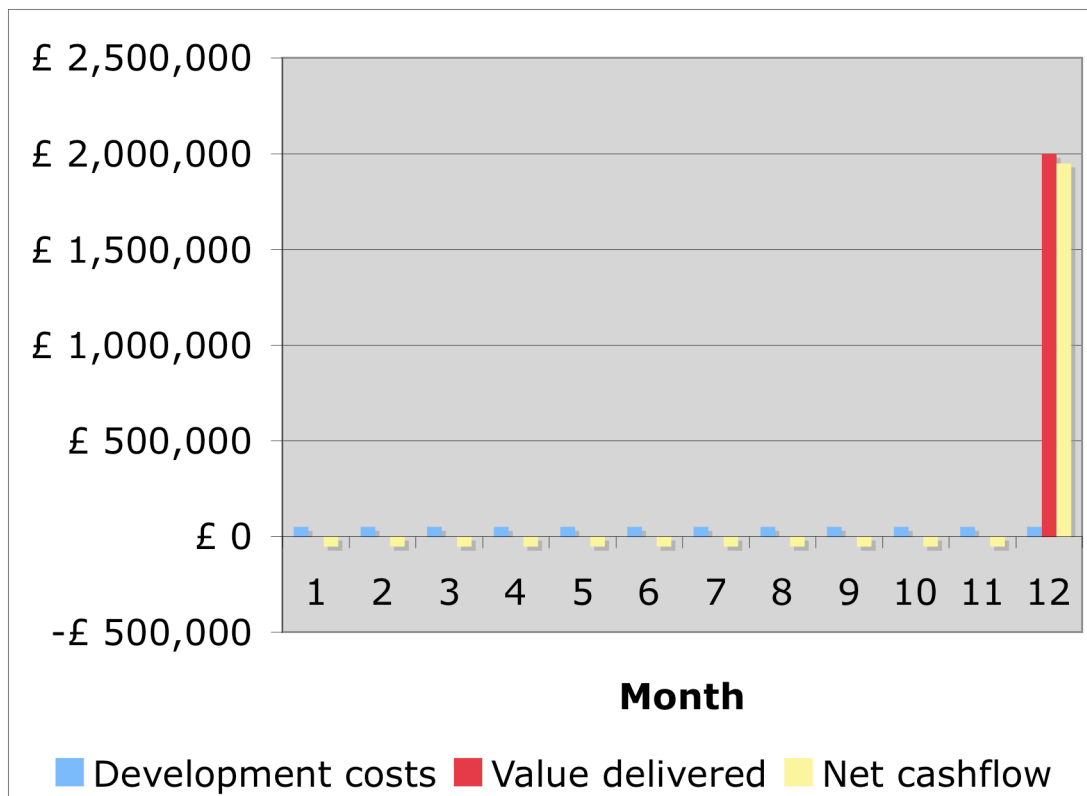


Figure 1 - Cashflow on a traditional Waterfall project

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 prioritise 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 some benefits will still accrue because something has actually been delivered. In a traditional project closing the project early usually only leaves part implemented, buggy, software of little value to anyone.

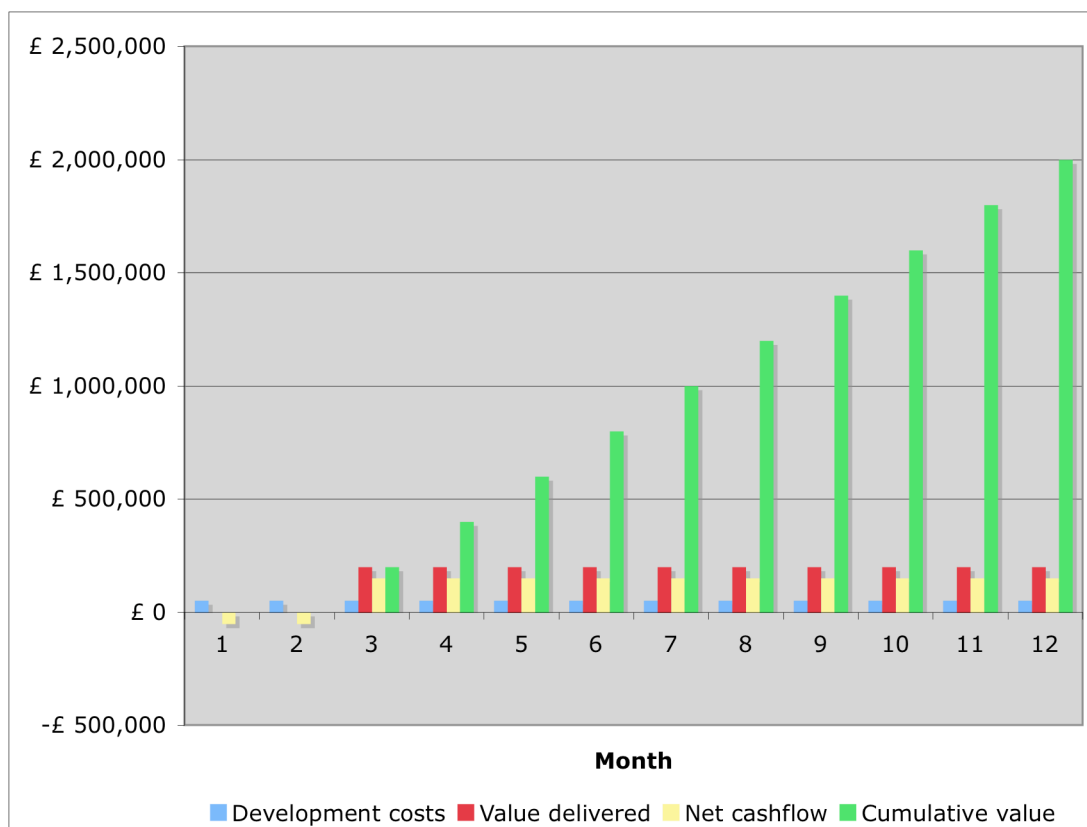


Figure 2 - Cashflow for an Agile project

Both Figure 1 and Figure 2 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 an 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%.

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 fixed analysis and then design can proceed prior to coding starting. 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 by adopting a *just in time* approach to analysis and design.

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.

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.

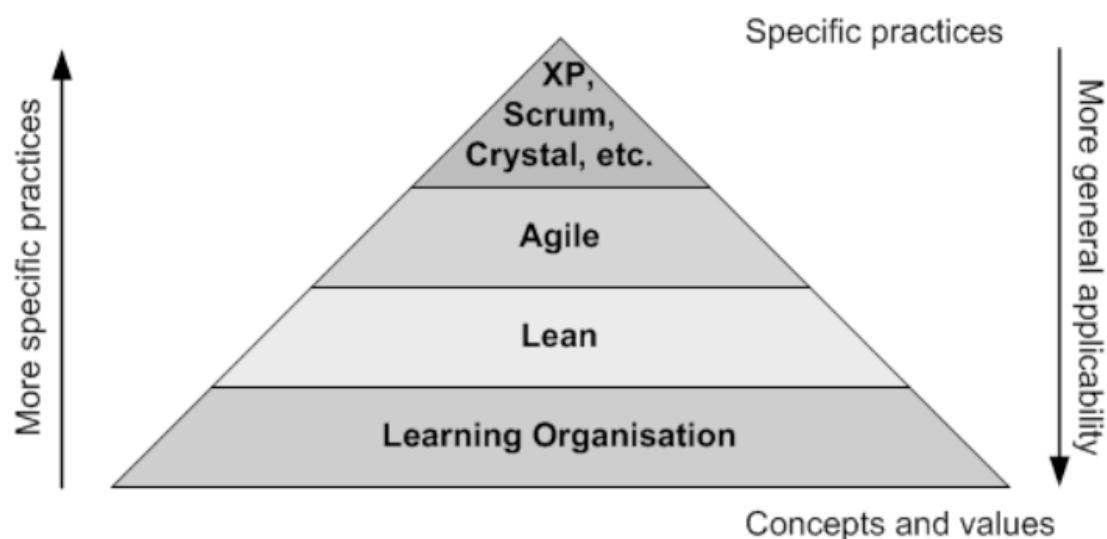


Figure 3 - How the Agile methods fit together

Agile as a whole is both an umbrella term to group all these methods – which were originally called light-weight 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 organizations 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.

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 the same principles to product design and development (Cusumano and Nobeoka 1998; Kennedy 2003) while Mary and Tom Poppendieck have applied the ideas directly to software development (Poppendieck and Poppendieck 2003; Poppendieck and Poppendieck 2007). Naturally, few of the specific practices of Lean manufacturing have made it software development.

Finally, underpinning all of these techniques are the ideas of *Organizational Learning* and *the Learning Organization* (Argyris and Schön 1996; de Geus 1997; Kelly 2008; Senge 1990). 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 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.

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. 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 prioritisation.

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 priorities 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 Agile methods adopt a *quality if free* approach and institute a number of techniques for boosting quality throughout the development cycle. As a result the test-fix-test cycle is broken. 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.

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 - is not intrinsically Agile. An Agile project may use, or even deliver, SOA services but the use of SOA on a project does not make a project Agile by itself.
- MDA – Model Driven Architecture – is not by itself Agile. As with SOA, an Agile project may be an MDA project, or it may build on MDA work but using MDA alone will not make a team Agile.
- Virtualisation: server virtualisation is useful tool for IT operations groups but it has little to do with application development. Used well it may well may IT operations for Agile but for development teams it is just another tool which may be useful.
- 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 a 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.
- Chaos: Agile development is not an excuse of project chaos or for engineers taking control of a project and refusing to listen to others.

Failures

Agile is not 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 do it re-risk projects and increase 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:

- **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, adopt test driven development or any other practice they dislike.

- **Potemkin Agile:** occurs when an a team adopts and applies an Agile method well but does not deliver business value. This is a form of goal deferment were the team consider *adhering to the process* rather than delivering business value as the success criteria.
- **Customer (Business Analyst, Product Manager, Product Owner) overload:** in 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 often bring in consultants and other experts to Agile enable a team. Some teams return to their old ways of working when the consultants leave. While advisers and consultants can be a great help when introducing Agile 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 maximise 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. Initial Agile adoption might start like but over time other groups fail to adopt. If they do not the Agile team will find it difficult to be truly Agile. When other groups do 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.
- **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 evangelise 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.

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 has experience both as a software developer and a development manager – largely with independent software vendors. He 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.

He is a trained product manager and project manager – holding PRINCE2 Practitioner status – in addition to his a BSc in Computing and management MBA.

Allan currently works as a consultant and trainer helping companies organise their software development activities and adopt Agile methods and is a partner in London Software Partners, <http://www.londonsoftwarepartners.com>. 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
Blue-White-Red	An example Agile system by the author which combined elements of Scrum with XP (Kelly 2007, 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.
Customer (Onsite Customer, Product Owner)	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. 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 "timeboxed" in that they have a defined start and end, and all iterations are the same length. Work is then sized to fit the iteration timebox.
Lean	Derived from the Toyota Production system as described in "The Machine that changed the world" by Womack, Jones and Roo.
Lean Software Development	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

	<p>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.</p>
Scrum	<p>An methodology developed by Ken Schwaber and Jeff Sutherland. "Scrum" does not stand for anything, it is a reference to game of Rugby.</p> <p>The Scrum Alliance certifies Scrum training in the areas such as Scrum Master and Scrum Product Owner.</p> <p>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).</p>
Scrum Master	<p>Scrum defines a new role of Scrum Master which is designed to help the team over come obstacles an improve. In part the Scrum Master is an Agile Coach. While many organization 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.</p>
Sprint	<p>From Scrum: an iteration, or a collection of several iterations which make up a release.</p>
TDD (Test Driven Design, Test First Development, Example Driven Design)	<p>Test Driven Design – originally part of XP, now a technique widely used in its own right.</p>
XP	<p>Extreme Programming – a methodology from Kent Beck with Ward Cunningham and Ron Jeffries. XP was initially the leading Agile methodology. This position has now been assumed by Scrum.</p>

References

- Argyris, C. and D.A. Schön. 1996. Organisational Learning II: Addison-Wesley.
- Beck, K. 2000. Extreme Programming Explained: Addison-Wesley.
- Beck, K. and C. Andres. 2004. Extreme Programming Explained: Embrace Change. Second Edition: Addison-Wesley.
- Cusumano, M.A. and K. Nobeoka. 1998. Thinking Beyond Lean: Free Press.
- de Geus, A.P. 1997. The Living Company: Nicholas Brealey Publishing.
- Kelly, A. 2007. "Blue White Red - an example agile process." ACCU Overload(81).
- 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. and T. Poppendieck. 2003. Lean Software Development: Addison-Wesley.
- Poppendieck, Mary and Tom Poppendieck. 2007. Implementing lean software development : from concept to cash. London: Addison-Wesley.
- Senge, P. 1990. The Fifth Discipline: Random House Books.
- Womack, J.P., D.T. Jones and D. Roos. 1991. The machine that changed the world. New York: HaperCollins.