The Agile Pyramid and the Learning View

By Allan Kelly, allan@allankelly.net, http://www.allankelly.net

The Agile Pyramid illustrated another, not incompatible, way of considering Agile and places it in relation both to specific Agile methods like Scrum and XP, and shows Agile's relationship to some other related theories.

The pyramid is not itself another perspective on Agile and it is compatible with the perspectives outlined earlier. Rather the pyramid situates Agile in relation to methods and concepts.

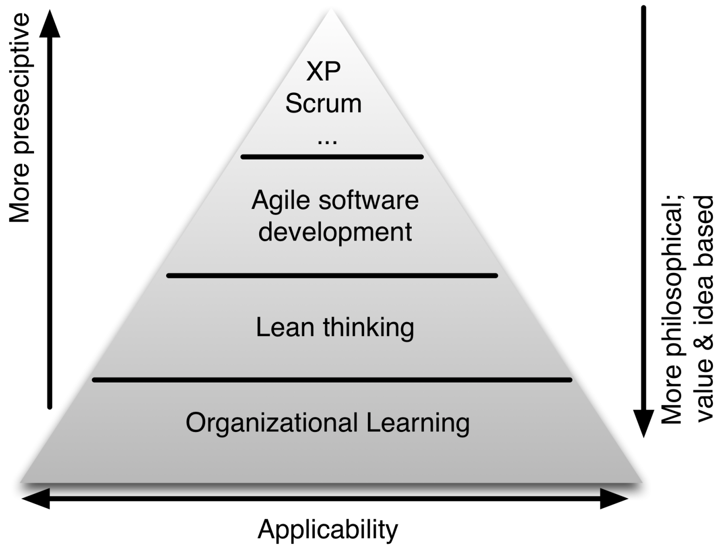

Figure 1: The Agile Pyramid

Near the top of the pyramid are more prescriptive approaches, conversely at the bottom are the less prescriptive, more value and principle based methods. Consider Extreme Programming. In the first edition of Extreme Programming Explained Kent Beck says:

> "The full value of XP will not come until all the practices are in place. Many of the practices can be adopted piecemeal, but their effects will be multiplied when they are in place together." (Beck 2000)

XP, especially in the first edition, is very prescriptive, you were encouraged to adopt all of it. It is clear on what you should be doing. While there are principles and values anyone reading the first book would be in little doubt as to what they should do.

Scrum proponents are often at pains to say "Scrum is not a methodology, it is a framework." Yet in practice teams are advised to "Do Scrum by the book, when you get good then modify it." In response to problems teams can be advised to "Do Scrum".

One of the apparent paradoxes in Scrum is that while teams are encouraged to inspect and adapt, learn and change the framework itself is off limits. Teams are not allowed to change the framework.

This author once observed a well know Scrum advocate. Faced with the question: "What do project managers do on a Scrum team?" the answer was "If you have a project manager on the team you are not doing Scrum."

In its own way Scrum is as prescriptive as XP. In fact most of the so called "Agile methodologies" exhibit these traits. To some degree this is as one would expect. The are very methods that promise to help you do software development. If they did not explain how to do this they would not be fulfilling their objective.

Agile software development as a whole is different in two ways. Firstly, as already mentioned, Agile itself is not so much a method as a toolkit. This Agile cannot be as prescriptive because one selects the tools to be used. Once selected though these tools may be prescription, e.g. stand-up meetings should happen at the same time every day. But teams are free to leave other tools unused.

Secondly, the term "Agile", and particularly "Agile Software Development" was applied retrospectively to describe and group a set of methodologies previously called "lightweight." Agile represents the common elements of multiple methodologies. (See http://agilemanifesto.org/history.html for the official history.)

And Lean

Close examination of Agile tools and methods reveals a similarity to Lean. The ideas behind Lean manufacturing (Womack, Jones, and Roos 1991) and Lean product development (Mogran and Liker 2006) - collectively "Lean thinking." If you haven't come across Lean before it basically means "The way Toyota do things." Toyota are not perfect but most of the time they are significantly better than their competitors.

Lean ideas such as: high quality, worker empowerment, just-in-time creation or delivery, feedback loops and visual tracking and others occur again and again in Agile. Personally I find it hard to find any idea or tool in Agile which cannot be traced back to the ideas contained in Lean. There are those who would argue that Agile is different from Lean but for me there are so many common points it is difficult to accept this argument.

Ward Cunningham, one or the originators of XP and Agile has stated (in a Twitter conversation) that when XP was being created the creators did not build it from Lean. While I have no reason to doubt him it is true that many Lean ideas (e.g. just in time manufacturing, waste elimination, etc.) were becoming well known during the 1990s. In the UK news reports of the new Nissan, then Toyota and Honda factories often reported these strange new ways of working.

Indeed both Scrum and XP, although less-so, directly draw on the work of Nonaka and Takeuschi (Takeuchi and Nonaka 1986; Nonaka and Takeuchi 1995). These authors studies Japanese industry (Honda, NEC, Matsushita, Canon and others). Even if the did not study Toyota they studied companies that adopted similar ideas. Many of these ideas can be traced back the work of W. Edwards Deming.

In other words: Ward Cunning, Kent Beck, Jeff Sutherland and the other originators of Scrum and XP may not have directly taken lessons from Toyota but the ideas were not difficult far away.

This author was lucky enough to here Professor Dan Jones, one of the originators of the term Lean, address the XP Day conference in 2008 - an Agile software development conference in London. Professor Jones directly address the relationship between Agile and Lean by saying that he saw no difference. He said he had examined "Agile software development" and he saw Lean. He didn't care what practioners called it, it was Lean to him.

While there are specific tools in Lean (e.g. A3 reports and value stream mapping) it is more of a philosophy of approach than a process prescription. The tools in Lean aim at helping practitioners improve their processes and environment. Principles like "one piece flow" and "eliminate waste" aim at changing and improve what already exists, or at designing new processes, rather than replacing one process with another - as, say, Scrum does.

Lean therefore is less of a prescriptive process and more of systems thinking model. I sometime think of Lean as a meta-method: while XP and Scrum tell you what to do, Lean tells you how to think so that you may decide what to do.

If there is a difference between "Agile" and "Lean" it may be that while Agile - true to its name - focuses on adaptability and change, Lean focuses on effectiveness and efficiency. Ultimately Lean teaches that building the wrong thing is waste and should be avoided, thus building the wrong thing because the need had changed is also wrong. So adapting and changing should be valued as a means to avoid waste.

The Learning Organization

Moving down the pyramid not only moves from prescription to principle but also widens the applicability. It is not possible to assemble a car with XP but it is possible to both design and assemble a car with Lean. Lean itself is a form of organizational learning, in order to become truly lean teams, departments and companies need to become learning organizations.

Peter Senge, one of the leading writers on organizational learning, described them as:

> "organizations where people continually expend their capacity to create the results they truly desire, where new and expansive patterns of thinking are natured, were collective aspiration is set free, and where people are continually learning how to learn together." (Senge 1990)

Learning organizations can be found in fields as diverse as oil production (e.g. Shell is an often cited example) and military offense (learning practices in the US Marine Corp are well documented.)

Friends and Family

The Kanban method might be considered at either the highest point - it is a method akin to XP or Scrum - or in the Lean layer. Kanban roots most of its theory and practice directly in Lean rather than Agile. Some Kanban proponents have even been known claim Kanban is not an Agile method. However for most practitioners Kanban is a member of the Agile family.

For these reasons I usually add Kanban as an vertical slice starting in Lean, passing through Agile software development, and ending in the top section of the pyramid - shown in the diagram below.

Not all methods can be easily categorized in this model. For example, the Crystal methodology (Cockburn 2002) is closer to organizational learning principles than other Agile methods.

Rational Unified Process - and is sibling Open Unified Processes - are not on the diagram either. RUP proponents claim RUP is, or can be, an Agile method. While I personally have never seen it used as such from what I understand about RUP it can be tailored to work much like XP or Scrum in short iterations.

RUP documentation is anything but Agile. A quick look at the IBM RUP website reveals this boast:

> "Leverage a diverse set of method content, guidance, templates, and processes with more than 100 selectable and customizable process best practices that can be applied to a variety of processes and domains." `http://www-01.ibm.com/software/awdtools/rmc/library/` 8 March 2013

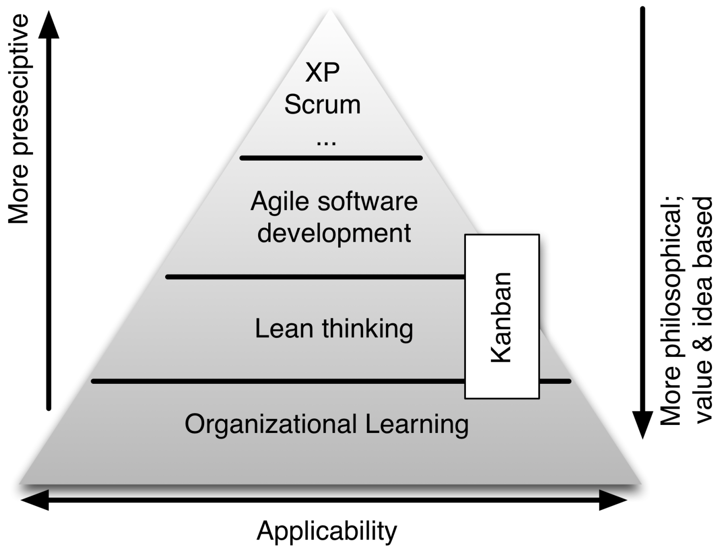

Figure 2: The Agile Pyramid

I cannot see how any method that contains over 100 customisable processes can be considered Agile.

Tom Gilb's Evo method (Gilb 2005) is an interesting case. Evo shares its roots with Lean but not in Lean, i.e. both build on the work of Deming directly - including the Plan, Do, Check Act cycle.

For the sake of simplicity I include "Systems Thinking" - and even "Soft Systems Thinking" in the Organizational Learning layer. Purists might argue with this but it keeps the model simple.

Similarly the model doesn't include several other related ideas - Dave Snowden's Cynefin model or Stephen Denning's work on story telling. While both might argue with placing these ideas in the bottom layer that is where they sit.

Devising a model that would include - to each authors satisfaction - would probably result in a model of such complexity that it would defeat the objective. For this model "Organizational Learning" is defined in very broad terms and goes beyond the Senge, de Geus and Nonaka.

Learning View

> "We understand that the only competitive advantage the company of the future will have is its managers' ability to learn faster than then their competitors." (de Geus 1988)

Those who study Lean in depth advocates the creation of a learning culture in a learning organization, e.g. (Mogran and Liker 2006; Shook 2008). Such cultures have long been studied by academics and senior business people alike, e.g. (Senge 1990; Argyris and Schön 1996; Tichy 1993), under the label "Organizational Learning".

At this level there is very little prescriptive advice. One of the best know names in the field, Ikujio Nonaka has written:

> "In the accumulation of over 20 years of studies, they [organizational learning writers] have not developed a comprehensive view on what constitutes 'organisational learning'." (Nonaka and Takeuchi 1995)

Organisational learning is almost entirely philosophical, value and principle based. Writers like Senge do give specific advice but this is the exception rather than the rule. Still, there are practices which can be adopted.

The concept of learning and change is key to even the most prescription Agile methods. Both XP and Scrum include the practice of the retrospective which is a formal team reflection and learning exercise. But learning is not confined to dedicated activities

All software development is an exercise in learning and knowledge management - as argued in the authors 2008 work Changing Software Development (Kelly 2008). The process of creating software involves learning in three domains - shown in:

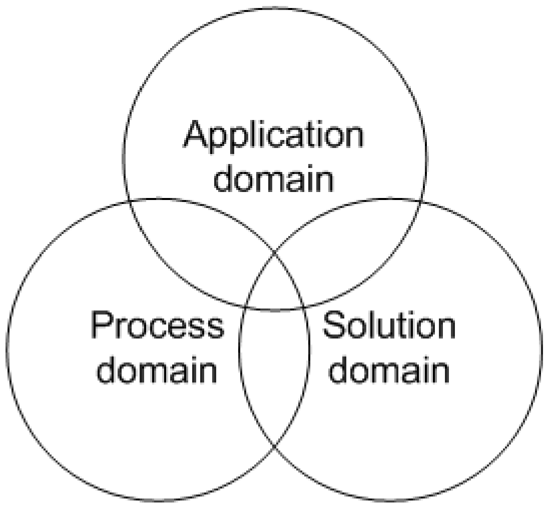

Figure 3: Three knowledge domains

- Application domain: both understanding the business problem(s) for which we trying to craft a solution and understanding what the solution needs.

- Solution domain: the tool and technologies we have available to build a solution, e.g. Java, PHP, Linux, etc.
- Process domain: the processes used to combine these things into a working, and valuable, solution

Traditional development, epitomised by the waterfall model, assumes that not only can the development process be segmented into several consecutive activities (e.g. requirements, design, coding, etc.) but that on completion of each activity all the required knowledge is known and learning can be frozen.

A second common assumption is that all the required knowledge is explicit and can not only be captured in written form but can be communicated through words alone; i.e. the writer will both write everything that needs to be written and the reader will fully and accurately comprehend that which is written. Consequently those responsible one activity can be released from their assignment once the documentation is completed.

This assumption, and the consequences demonstrates a lack of understanding of both tacit knowledge and knowledge sharing norms (Davenport and Prusak 2000; Nonaka and Takeuchi 1995). Once one understand the role of tacit knowledge and the difficulties in transferring knowledge from one person to another the waterfall model ceases to make sense.

Instead if one considers software development as a learning activity occurring in multiple domains simultaneously over time it becomes clear that it is simply not possible to characterise development as a set of hermetically sealed modules. (No matter how attractive that idea is to the logical brain.)

Under the learning view we learn a little, we act on the learning, we learn from the action and the process repeats. In other words: there is an iterative cycle involved. Deming's Plan-Do-Check-Act is a formalisation of this learning process.

If we look at the software development process there are at least four key learning activities:

- We learn new technology.
- We learn the application domain.
- We problem solve by applying our technology knowledge to the application domain.
- Users learn to use our application and learn about their own problem, which changes the application domain

Each one of these points reinforces the others: in our effort to solve a problem we need to learn more about the problem; our solution may use a technology that is new to us. When the user sees the end product their mental model of the problem will change too. They too will learn, through the software, and acquire new insights into the task that may lead to changes to the software.

Learning is thus inherent at every stage of software development. We can either choose to ignore it and muddle through somehow, or to accept it and help improve the learning process.

The implications of this view go beyond the development of software. Deployment, introduction and use are all themselves learning activities. It is this learning that leads to requests for enhancement and change. The only software which does not receive such requests is software which is not being used, where no learning is occurring.

Unlearning

Learning Agile is the easy bit, the difficult bit is unlearning past behaviours and practices. This is especially true when behaviours have been associated with success in the past. Such success might be ones own experience or it might be passed on through formal training.

When first adopting Agile there is little danger in continuing old habits - perhaps maintaining Gantt charts and Burn-Down charts. However if these habits persist then organization will effectively have two approaches in parallel. At the very least this will increase cost, but more troubling it also opens opportunities for tension and conflict. What if the Gantt chart says a target date will be best the Burn-Down says not? Maintaining old practices in parallel will also negate the benefits of Agile practices. For example, if a team has fully automated unit and acceptance tests running is there still a need for manual system testing?

In the beginning, when a team if first adopting automated testing, it makes sense to keep the traditional manual tests. Once the automated testing is adequate systems tests are not only expensive but they take time. This time is usually taken when the system is nearing release. Thus a manual test phase delays delivery.

Unfortunately things like manual system test phases can be a “comfort blanket” for organizations. Perhaps harmless, or perhaps a block to maturing and advancing; worse the presence of a manual safety net may cause some not to invest as much effort in automation. Gantt charts and system test are but two examples of practices which need to be unlearned. Such practices and beliefs may once have made a positive contribution to success now need to be unlearned if a team is to reach full potential.

Pyramid as an Adoption Model

There is another way of view offered by the pyramid model. It is possble to view the pyramid itself as a change model, shown in the next figure.

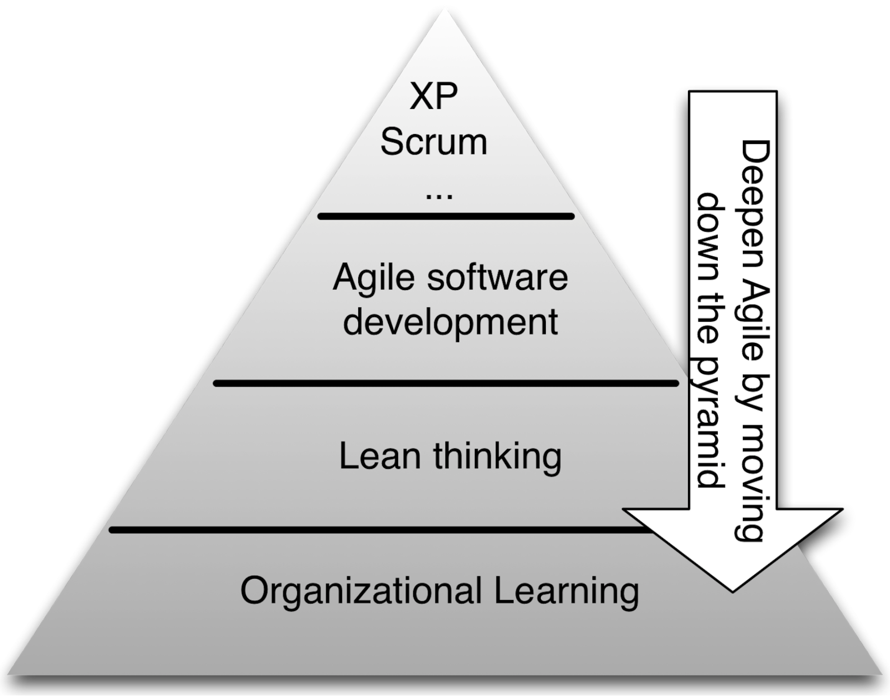
XP
Scrum
...
Agile software development
Lean thinking
Organizational Learning
Deepen Agile by moving down the pyramid

Figure 4: Deepening Agile, creating a Learning Organization

Starting with a well defined Agile method and proceeding in this fashion makes logical sense. Some companies and teams start by adopting an Agile method, say Scrum out of the box.

Over time, if successful the team will add more tools and techniques from the wider Agile toolkit. Given more time, more reflection and more experience, ideas from Lean are likely to permeate the teams thinking and behaviours. Ultimately the team, and company, are creating a learning organization.

Learning can stall at any stage, learning disabilities can prevent the team from initial adoption or improving and moving down the pyramid. For example: companies that mandate processes or demand common process across teams limit change to the pace of the slowest team at best.

Again, the difficult bit of this adoption process is the need to unlearn. For example: teams which have experience success with strict Scrum sprints may be reluctant to vary length or give them up, such reluctance may blind them to opportunities for levelling flow.

Arguably the emergence Kanban itself is the result of this deepening process. And possibly some of the resistance to Kanban shown by experienced practitioners is itself a reluctance to unlearn.

Innovation

A further dimension which need to be consider is that of innovation. Innovation is an important part of software development both at a micro level - finding new coding solutions to new and old problems. And at macro level software underpins innovative solutions upon which businesses are based.

Quite simply, innovation cannot be scheduled or planned for. It is not possible to draw up a project schedule with dates label “Innovation happens here”.

Innovation is both the result of learning and a source of learning. Similarly innovation creates change and change leads to innovation. Finally, true learning only occurs when one acts on learning, i.e. change occurs, and again, change creates learning. Innovation, learning and change are all part of the same process - show below.

Conclusion

Agile, like Lean, is built on learning. Learning activities are inherent in all Agile methodologies even if formal learning is sometimes absent from daily practice. However, unlike traditional development Agile allows for learning rather than trying to free it out of the process - and thus freeze out change and innovation.

Overlapping activities not only provides a more accurate model for how software development happens (because it is a learning activity) but also opens

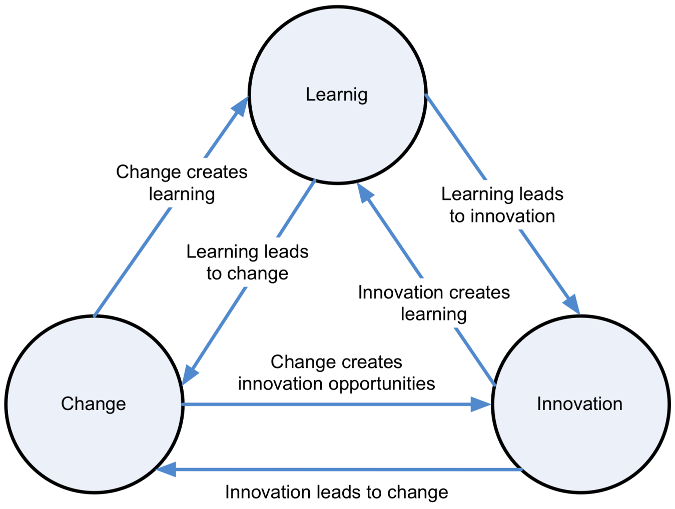
Learnig
Change creates learning
Learning leads to innovation
Learning leads to change
Innovation creates learning
Change creates innovation opportunities
Change
Innovation
Innovation leads to change

Figure 5: Learning, Innovation and Change

opportunities for more learning. Teams which only ever work in the Iterative Style of Agile restrict these opportunities because they continue hold the initial requirements as set.

Agile, and the Agile methods, are fellow travellers with other theories, concepts and models which all build in a circular, usual learning driven, cycle.

Building an Agile company is really a question of building a Learning Organization. Learning needs to happen in multiple domains and at multiple levels. The longer an organization is on this journey, and the deeper they progress down the pyramid, the more they need to discover and invent for themselves. The Agile toolkit provides many good tool to get a team started but in time they must think, learn, change and innovate for themselves.

Humans are natural learners. The first step to building a learning organization - and adopting Agile - is to remove the barriers that prevent learning. Resolving learning disabilities is necessary but may not be sufficient alone.

References

Argyris, C., and D. A. Schön. 1996. *Organisational Learning II.* Addison-Wesley.

Beck, K. 2000. *Extreme Programming Explained.* Addison-Wesley.

Cockburn, A. 2002. *Agile Software Development. The Agile Software Development Series.* Addison-Wesley.

Davenport, T. H., and L. Prusak. 2000. *Working Knowledge.* Harvard Business School Press.

de Geus, A. P. 1988. “Planning as learning.” *Harvard Business Review* 66 (2): 70.

Gilb, T. 2005. *Competitive Engineering.* Butterworth-Heinemann.

Kelly, A. 2008. *Changing Software Development: Learning to Become Agile.* John Wiley & Sons.

Mogran, J. M., and J. K. Liker. 2006. *The Toyota Product Development System.* Productivity Press.

Nonaka, I., and H. Takeuchi. 1995. *The Knowledge Creating Company.* Oxford: Oxford University Press.

Senge, P. 1990. *The Fifth Discipline.* Random House Books.

Shook, J. 2008. *Managing to Learn.* Cambridge, MA: Lean Enterprise Institute.

Takeuchi, H., and I. Nonaka. 1986. “The New New Product Development Game.” *Harvard Business Review.*

Tichy, N. M. 1993. “Revolutionize your company.” *Fortune* 128 (15). http://money.cnn.com/magazines/fortune/fortune_archive/1993/12/13/78732/index.htm.

Womack, J. P., D. T. Jones, and D. Roos. 1991. *The machine that changed the world.* New York: HaperCollins.

About the author

Allan Kelly has held just about every job in the software world, from system admin to development manager. Today he works as consultant, trainer and writer helping teams adopt and deepen Agile practices, and helping companies benefit from developing software. He specialises in working with software product companies and aligning products and processes with company strategy.

He is the author of two books “Business Patterns for Software Developers” and “Changing Software Development: Learning to be Agile”, the originator of Retrospective Dialogue Sheets (http://www.dialoguesheets.com), a regular conference speakers and frequent contributor to journals.

Allan lives in London and holds BSc and MBA degrees. More about Allan at http://www.allankelly.net and on Twitter as @allankellynet (http://twitter.com/allankellynet).