Lean and Agile in a Nutshell

2

Create Value, Eliminate Waste, and Adapt to Change

Agile and Lean

In Chap. 1, we started to develop the concept of "management" and "leadership" that you can read on the cover of this book. Now is the time to go into the "Agile" part of it.

In this (small) chapter, the basis of Agile and Lean will be reviewed to establish a common understanding and framework that allows further discussions. This book does not intend to be a comprehensive guide of what is Agile and Lean, nor a guide on how to implement Agile/Lean in your company, but I'll try to highlight the main implications of this concept for the organization and not only at the team level.

Again, this is not a book about Agile or Lean (oh my, I would need several books to explain those concepts deeply). But you can find some other excellent books on that topic out there. Some of the ones I recommend are those by Mary and Tom Poppendieck (Lean Software Development Series), Mike Cohn (specially the brilliant *Succeeding with Agile*), and Henrik Kniberg (*Scrum and XP from the Trenches*). And of course you can always engage in plenty of seminars and courses on the topic.

As many people (me included) consider that Agile is partly an evolution of Lean into the field of software/product development, I'll start with some background on Lean production.

Aichi Prefecture, Japan, 1946. . .

Á. Medinilla, *Agile Management*, DOI 10.1007/978-3-642-28909-5_2,
© Springer-Verlag Berlin Heidelberg 2012

The Japanese Revolution

In 1946, Toyota was a rather small car manufacturing plant in a country that had been devastated by World War II. The usual raw-material supply problems Japan had always experienced because of the fact of being an island in the middle of the Pacific Ocean were worsened by this situation, and the Americans transforming their mighty war production machinery into goods manufacturing plants were not helping, as they needed to sell those goods all around the world and were flooding markets with them (which explains the Marshall plan and other strategies for rebuilding Europe, but that is another story).

Furthermore, the country's morale was almost destroyed. After two atomic bombings, 3 million dead, 9 million homeless, the losing of the colonies, and the surrender of the nation by the Emperor, one would have expected the proud sons of the Samurai to commit collective ritual suicide or *Seppuku*.

But this is not what happened. Not at all. The Japanese reacted to this situation by bootstrapping the nation and devoting their millenary discipline to transform it as they had already done during the modernization in the Meiji period. This postwar rebirth of Japan was called the Showa era (1946–1989), and it has been repetitively referred to as "the Japanese miracle": building one of the most economically powerful countries of the world out of ashes and people's will. To address the magnitude of the change, realize that per capita income in Japan rose ten times between 1950 and 2008. And Toyota, for instance, became the world's biggest car manufacturer by 2007.

In fact, more than secret techniques, tools, or business plans, it was actually the collective effort and motivation of the workforce inspired by their leaders that made the whole transformation possible.

Management guru John Kotter dedicated several of his writings and studies to the figure of Konosuke Matsushita, and in his opinion, the start of the Japanese miracle can be traced back to the moment when, at the end of the war, he addressed his demoralized workforce and said, "I have been thinking about purpose." He then described a company that would force competitors to produce at the same outstanding levels of quality, innovation, and low prices that they will achieve, thus eliminating poverty in Japan and creating "a paradise on earth in the span of the next 250 years." It is been reported that several employees stood up in tears and said, "I think I could dedicate my life to this."

But let's go back to Toyota. By a similar process of continuous improvement and collective effort like the one going on in Matsushita, or even in the whole country, they designed a production system that was later studied by the Americans and baptized "Lean production." This system was based on a simple set of principles and rules that, when embraced by an engaged and empowered workforce, produced a plethora of tools, techniques, practices, artifacts, processes, roles, etc.

Unfortunately, for a long time, western companies tried to replicate those without really understanding the principles, which led to the rise of a set of approaches like Six Sigma, process reengineering, total quality, and other fads that tried to mimic the Japanese ways in a long-lasting, painful (and profitable) way of "Cargo Cult."

> "In the South Seas there is a cargo cult of people. During the war they saw airplanes land with lots of good materials, and they want the same thing to happen now. So they've arranged to imitate things like runways, to put fires along the sides of the runways, to make a wooden hut for a man to sit in, with two wooden pieces on his head like headphones and bars of bamboo sticking out like antennas – he's the controller – and they wait for the airplanes to land. They're doing everything right.
>
> The form is perfect. It looks exactly the way it looked before. But it doesn't work. No airplanes land. So I call these things cargo cult science, because they follow all the apparent precepts and forms of scientific investigation, but they're missing something essential, because the planes don't land."
>
> –Richard Feynman, *Cargo Cult Science*, 1974 Caltech commencement address

If you ever want to be successful on a Lean approach, please take your time to understand the basic rules and principles because those are the ones that you really have to plant on your soil: very probably, they will become some different kind of crop from the one the Japanese are harvesting, but it will be the one that suits your needs.

> "Copying practices without understanding the underlying principles has a long history of mediocre results. But when the underlying principles are understood, it is useful to copy practices that work for similar organizations and modify them to fit your environment."
>
> –Mary and Tom Poppendieck, *Lean Software Development*

The Five Principles of Lean

The whole Lean universe can be summarized in as few as five basic principles. Easy, right?

The first one is *understand and maximize value*. Very often, Lean is described as a way of "eliminating waste," but this is only a derivate of this principle. The hard part is trying to see your whole system as a way to produce value for your customer and truly understand how you provide that value.

Not everything your client pays for is value, and on many occasions, companies are dead wrong about how their client sees the value of the product.

For example, at a meeting on a very well-known drill-machines manufacturer, the CEO asked the management board "Gentlemen, what it is that we sell to our clients here?" Everyone, as expected, replied, "The best possible, most powerful, cheapest, highest quality drill machines in the world." But the CEO replied, "nonsense! You don't understand this company! Holes! Holes in a wall is what we sell!"

The CEO understood a deep truth about what their client valued. If there was a better way of making holes in a wall than using a drill machine, the company was wasting its efforts on building better drill machines. Because this is not what their clients truly valued.

Once you understand value, you can start looking at the whole company and asking yourself "how is this activity adding value to our client?" It is a dangerous question to make to yourself because you will realize that many of the things lying around your plant are not directly contributing to your client's experience. And then, on a Lean environment, they have to be labeled as "waste" and you have to strive to reduce them to the minimum or even eliminate them.

For me, the magical question you have to make to identify waste is, "are we willing to do more of this? Like, for example, double it?" With this question, you'll rapidly realize that meetings, managers, reports, inventories, transportations, rework, handoffs, overproduction, delays, and many other kinds of "wastes" are things that you have to eliminate or reduce to the minimum – but not less than the minimum, of course.

> "Make things as simple as possible, but not simpler."
>
> –Albert Einstein

Many times managers feel discomfort when I label management as "waste," but be honest to yourself: if management is true "value" for your client, are you willing to double the number of managers on your company? Maybe triple it?

Do you prefer to fly with one company or another depending on which one has the most managers?

No, you want to have the minimum viable number of managers that works for your company. And then find the way to work with even less managers.

The second principle of Lean is *optimize the value stream.* That means understanding how the different activities of your company contribute to producing value and then finding the way to place them on a sequence that shortens the production cycle, hence reducing both cycle time (time to produce a feature from work start to end) and the lead time (elapsed time since the feature was actually asked for by the client).

A very popular tool to implement this second principle is value stream mapping. To implement a value stream map for your company, just take a pencil and some paper (experts stress the importance of not using software and doing this by hand) and start from the end, that is, the value delivery to your customer. Then start tracing back where this value came from, how much did it take, what effort was put into it, how much of this effort was actually waste, and how much was value-adding activities. Then repeat and continue until you arrive at the place where your client order was received. If you divide the value-adding time by the total time – including queues, delays, waiting periods, and non-value-adding activities – you'll have a performance ratio indicator of your process. For companies that haven't been through a Lean initiative, performance ratios of 9–20% are not unusual. If you do your first Value Stream Map and you obtain something above 50%, don't congratulate yourself too much – you have to look twice, possibly with the help of an expert.

Optimizing the value stream is not only a matter of mapping, measuring, and reducing: it is a crucial exercise to understand and see your company as a whole, not as independent black boxes each of them trying to do the best for themselves. As systems engineering teaches us, any attempt to optimize one part of the system will very probably suboptimize the whole.

> "The obligation of any component is to contribute its best to the system, not to maximize its own production, profit, or sales nor any other competitive measure. Some components may operate at a loss to themselves in order to optimize the whole system, including the components that take a loss."
>
> –Edward Deming, *The New Economics*

The third Lean principle is *pull production*. This means that new products are only manufactured when the client needs them, reducing the need of stocks, inventories, or in-excess production. Small stocks are maintained, and when one piece is consumed, another one is immediately produced to replace it. For this to work, the system must be able to change their production and deliver very fast to react to the needs of their clients, thus embracing change and uncertainty. Any attempt at "Manufacturing Resource Planning" or "yearly forecasts" is an anathema for a Lean industry.

Lean skeptics will argue that some kind of planning is needed, and that is absolutely true. But in a world where something like Facebook appears and makes 50 million clients in less than 2 years, there is a need to react faster than a year's time. Planning and executing cycles must be reduced to the minimum possible, and that is why most software industries in the Agile or Lean startup frameworks are moving to 2-week production cycles of plan, execute, release, and collect feedback.

For instance, the whole design and production of the Toyota Prius, a new concept hybrid engine car, took Toyota roughly a year, when the industry average for a new regular model of car was 4–5 years. And we are talking of 1997 here. Nowadays, the Spanish company Inditex is able to deliver clothes to any store worldwide in 72 hours, thus reducing the need of huge stocks at the stores and being able to react when some particular model is selling well by increasing the production of that particular model. The contrary is also true: if some model is not selling, the most you are losing is 3 days of production of that model, as there is no need to produce millions in advance without actually knowing if someone will even look at it.

For the third principle to work, the fourth principle is needed: *single-piece flow*, defined as the ability of a single order or piece of work to flow smoothly across the whole system without interruptions and at the maximum possible speed. This usually implies reducing the amount of things going on at a particular moment, as Little's law establishes that cycle time increases

with the amount of pieces on the system. In other words, the more things you think you are doing at the same time, the less productive you become.

This means that multitasking is a myth, as is the "total productivity." In his book *Quality Software Management: Systems Thinking*, Gerald Weinberg measured the performance of an individual working on several projects: compared to working with a single project, the loss of productivity because of context switching was as much as 40% for someone working simultaneously in three projects and 75% for someone at five projects! This is the reason why Kanban systems' main rule is to limit the work in progress (WIP), with an ideal goal of WIP limit "one," meaning that you don't start another task until this one is finished, even if it gets blocked. This idea of "focusing on the next actionable item" and not starting anything else until this one is finished is also one of the bases of the very popular "Getting Things Done" or GTD personal productivity and time management system by David Allen.

Finally, the fifth and my personal favorite Lean principle is *continuous improvement*, or in Japanese "*Kaizen*." – a hopeless but joyful strive for perfection that makes us better every day. Today, better than yesterday. Tomorrow, better than today. A martial state of mind that makes us train constantly and never be satisfied with our current skill, no matter how high it is.

> "This old man must still train and train"
>
> –O Sensei Morihei Ueshiba, Aikido founder, declared "Sacred National Treasure" of Japan, one of the greatest martial artists of its time, at the age of 86, shortly before he passed away.[1]

Again, Lean skeptics will try to argue that "perfection is impossible" or even say "why do we want to be better? We are just fine now…." But on a Lean environment, there is a perpetual state of discomfort with the current state. Lean leaders will constantly push the organization out of its zone of comfort in search of a new, higher level of quality, performance, and, overall, client satisfaction.

[1] Ueshiba M, Ueshiba K (1996) Budo: teachings of the Founder of Aikido Ueshiba. Kodansha International, p 21.

> "What's perfection good for if we will never reach it? It gives us a true north so we keep walking on its direction"
>
> –Anonymous banner at the 15-M revolts in Madrid, 2011[2]

The 14 Principles of the Toyota Production System

Dr. Jeffrey Liker presented a deeper insight on Lean production systems in his 2004 book *The Toyota Way*. He summarized 14 principles and behaviors that described Toyota's managerial approach to Lean production:

1. *Base your management decisions on a long-term philosophy, even at the expense of short-term financial goals*. There is always an urge somewhere, a fire elsewhere, and something critical to be done anywhere else, but you have to understand that there will always be. So if you start dropping trainings, holidays, market events, innovation workshops, or improvement programs because of delivery dates, you are investing in short-term goals and creating an ever-growing learning debt. In a Lean environment, provided we have enough resources to invest and provided this investment will not ruin the company in the short term, we will always try to invest in long-term improvement, learning, and growing, even at the price of short-term losses.
2. *Create a continuous process flow to bring problems to the surface*. Searching for new forms of waste and ways to reducing it shall not be limited to a certain number of "Kaizen events." A regular process must be put in place instead, and management must make sure that all problems detected by the workforce and tagged as "waste" or "impediments" are urgently addressed as national emergencies, even if it means delaying short-term projects (first principle).
3. *Use "pull" systems to avoid overproduction*. This was one of the five original lean principles and was so reflected by Liker in "The Toyota Way." The pull and flow principles led to the revolutionary "just-in-time production" concept, meaning that pieces and products were only delivered exactly when needed, thus reducing the need of inventories, warehouse space, and other forms of waste.

[2] As seen by myself.

4. *Level out the workload (heijunka).* Instead of bursts of work followed by periods of lower activity, try to achieve a sustainable pace and a constant production rate. Variability of demand can be absorbed by small production buffers (sometimes called heijunka boxes), producing in small batches (hence being able to adapt better to a changing demand) and achieving low die change times (being able to change between the production of two different pieces very quickly and at a low cost).

5. *Build a culture of "stop the line*," which means that when a problem is detected, it is more important to stop production and fix the problem, making sure that we will never make the same mistake again, than continuing the production and making a mental note of "we should be fixing this later on." Stopping the production line has a short-term cost, but the implication in the long term is huge. Maintaining a pile of defects, problems, impediments, and inefficiencies creates a "compound interest" effect on quality, thus creating a "technical debt." Some examples of "stop the line" culture are *Andon* devices, a red-light signal that tells management that a problem has been detected and the line needs to be stopped, and *Jidoka* or "autonomation," automation with a human touch, which means that the whole product line can be stopped automatically when a machine automatically stops and a human supervisor decides that the problem must be addressed. In software companies, these principles have been implemented in the form of automatic test, build, and deploy systems that tell the programmers when something went wrong during the automated process.

6. *Standardize work*, but not in the way western companies understood it during the 1980s and the 1990s. Toyota understood standardized work as a tool to make sure that everyone was doing things the same way, so when they changed something, they could measure the effects of that change and evaluate if it was a good idea or not. It was also a way for every worker to know what he was expected to do. But workers were responsible for constantly reviewing the standard in search of better ways to perform, and managers were accountable for it. It is said that Taiichi Ohno, considered one of the fathers of the Toyota Production System, fired a manager because the standard that his team was following had not been changed in more than a year. Please make sure you understand this: standards must change constantly. Otherwise we are considering that we have found the perfect way of working, which is by definition not possible, or we consider that this way is good enough for us, which is against the principle of continuous improvement.

7. *Use visual control so no problems are hidden.* Again, western companies will always come up with software tools, written reports, or electronic dashboards to try to know what is happening, but according to the general experience of Lean and Agile *Senseis*, nothing beats the simplicity of visual controls. Kanban cards, team boards, signs, banners, *Andon* devices, A3 handwritten reports, floor layout lines, or silhouetted tool boxes where you can easily know where to place everything are typical examples of visual management in a Lean manufacturing plant. Visual control devices also foster team collaboration, employee empowerment, and a sense of ownership that no electronic tool has been able to reproduce.
8. *Use only reliable, thoroughly tested technology that serves your people and processes.* Align technology with your process and not the other way round. Technology must help your people do things easier, faster, and in a more efficient way. If the technology you are using is making people hate it, you are probably serving some needs other than the one of your people and your process. Or your process is darn wrong!
9. *Grow leaders who thoroughly understand the work, live the philosophy, and teach it to others.* There is no better way to teach people than personal example. Lean leaders are a living incarnation of the Lean principles and values, who feel that their main responsibility is to make everyone understand the company's culture. Japanese Lean Senseis often say that the main problem with western managers is that they want to rule and command, not to teach.

> "We must become the change we want to see in the world."
>
> –Mahatma Gandhi

10. *Develop exceptional people and teams who follow your company's philosophy.* Again, the Lean concept around people is to make them responsible for the process. Empowerment and ownership are frequent terms when describing people's behavior in a Lean environment. As stated in the previous principle, Lean leaders are devoted not to command and control their people but to inspire the purpose, principles, and values that lead to the desired results.

> "If you want to build a ship, don't drum up the men to gather wood, divide the work and give orders. Instead, teach them to yearn for the vast and endless sea."
>
> –Antoine de Saint-Exupery

11. *Respect your extended network of partners and suppliers by challenging them and helping them improve*. The supply chain is not to be considered a zero-sum game where everything that your supplier is earning is something that you are losing. By developing a trust relationship with your partners and helping them improve, you are improving your own process, and joint ventures between suppliers and clients based on this kind of trust have proved to systematically produce amazing results and long-term win-win situations.
12. *Go and see for yourself to thoroughly understand the situation*. Often translated as "management by wandering," *Genchi Genbutsu* is an important practice for the Lean leader: don't rely only on reports, metrics, scorecards, and meetings. Go where the action is. Live with your team, sit with them, and experiment by yourself what is happening at the *Gemba*, the place where work is being performed. Help your team perform better and instruct them at the workplace. By the way, if you are not able to personally train your people, you are probably not qualified to be a true Lean leader and are working under the old Taylor paradigms of thinking managers versus working labor instead.
13. *Make decisions slowly by consensus, thoroughly considering all options*, and then implement decisions rapidly (*Nemawashi*). Incorporating everyone's opinion in the decision-making process can be hard, but once again this short-term investment produces great long-term results: a more committed workforce, better decisions based on a more complete view of the system, enhanced collaboration, and less conflicts.
14. *Become a learning organization through relentless reflection (Hansei) and continuous improvement (Kaizen)*. The master principle of Lean: never be satisfied with your current state, strive for perfection. Every mistake is seen as an opportunity to improve as long as people are empowered to take risks, make mistakes, and learn from them. Blame-avoiding games will never make the company better, they will only keep you safe, and trying to ignore mistakes is another form of blame avoiding. In a Hansei-Kaizen culture, everyone feels responsible and accountable for their decisions, and when a mistake has been made, it is

more important to fix it and make sure that no one else will make the same mistake again than trying to hide the broken glass under the carpet.

Over these 14 principles, constant improvement and respect for people remained as the foundations to correctly implement them, and this remains the base of Toyota's competitive advantage.

Kanban Systems

The five master principles (Value, Value Stream, Pull, Flow, Kaizen) and the "Toyota Way" 14 principles gave birth to a plethora of practices and tools that have been deeply covered by the Lean literature. We've already mentioned Jidoka, Andon, Heijunka, Value Stream Mapping, Genchi Genbutsu, Just-in-Time, Gemba Kaizen, A3 reports. . . And there is more: You can read about *Poka-Yoke* or foolproof design (e.g., the USB connector that can only be plugged one way); *5S plant maintenance systems* (Sort-Set in order-Shine-Standardize-Sustain); *Single-Minute Exchange of Die*, or the ability to change from one product manufacturing to another very fast; *or root cause analysis* as a way to look at problems, often combined with *Ishikawa fishbone diagrams.*

As you see, we would need a whole dictionary to describe all the tools and practices that the Lean enterprises have produced in the last decades as forms to implement the 5 root principles and the 14 Toyota Way principles. But one of them is especially important because of its relevance that has been recognized recently in the IT industry: *Kanban systems.*

> "The two pillars of the Toyota production system are just-in-time production and automation with a human touch, or autonomation. The tool used to operate the system is kanban."
>
> –Taiichi Ohno, *The Toyota Production System*

Modern Kanban authors like David J. Anderson make a difference between "kanban," a tool to manage demand, and "Kanban," with a capital "K," a framework to look at production systems, spot bottlenecks, and trigger employee empowerment and continuous improvement.

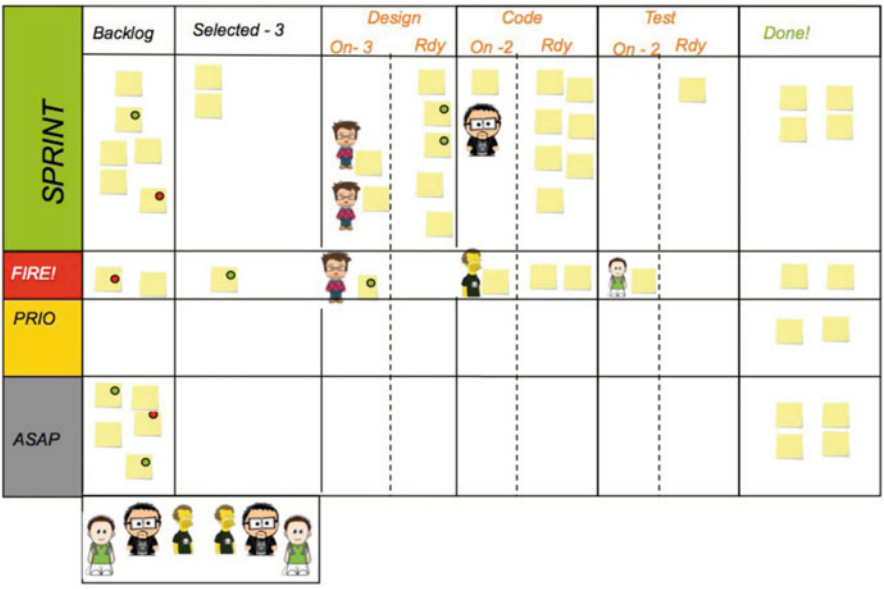
The original idea behind Kanban systems is to attach a physical card – or kanban – to any piece of work being done at the factory and then limit the amount of available kanbans to match the production capacity. Thus, any order coming when no kanbans are available has to wait on a buffer until

some resources are released. If demand is higher than capacity, the buffers will start to grow, and this will be understood as a need to rearrange the system so it is able to cope with the growing demand.

This can be achieved by adding more resources, of course, but it is usually possible to raise the capacity also by improving the production process, eliminating waste, removing impediments, training people, improving quality... Companies that directly add more resources whenever they have a peak of demand have ever-growing costs, and their quality remains the same or worse, while companies that strive to improve their system reduce their costs, raising their production capacity and their product quality.

Kanban systems also provide a simple way to "see the whole" by gathering all the kanban cards at a given moment and putting them on a board that represents the production process. Measuring the time a kanban takes to move from one production step to another, we can obtain valuable information on cycle time (the time a single unit takes to be produced since we start working on it) or lead time (same, but counting from the time the order entered the system). We can also see where the kanbans are stopping and growing queues, or even where a production cell is running out of work.

For example, look at this simple kanban board for a software development team of four people. Every kanban represents a piece of work that has to be designed, coded, and tested, so the flow of work is represented from left to right. Small comic-like avatars are also used to represent the people working at them:



It is easy to see that the poor guy on the test column has a lot of "code ready" kanbans to work on. Of course we could add more testers. But maybe the problem is that the "code ready" code is not as ready as the coders think, so maybe we should invest our resources better by training the coders and asking them to spend more time on their work until the quality of the code is enough for a smooth testing. This would have both the effects of reducing the throughput of the coders and making the tester work faster, as he would have better quality materials to work at. So we would have equalized the system without introducing new costs in the form of more – demoralized – testers. As you can see, the visualization of the value stream and the work in progress help us understand the whole and improve the system.

A true Kanban system works with a simple set of rules:

1. Start by doing exactly what you are doing right now.
2. Map the value stream.
3. Visualize all work on the value stream by adding kanban cards.
4. Introduce work-in-progress (WIP) limits.
5. Help the system flow and improve everything.

The WIP limits are usually one of the most difficult steps when implementing a Kanban system. We have been trained on the myth of "multitasking": doing several things at the same time as a way of being more efficient. But, in fact, reality works the other way round: the less things you are doing in parallel, the more efficiently you will perform them. When you switch between two or more tasks, there is an inherent loss of capacity due to context switching, and the more things you are switching between, the longer it takes to finish every single one of them.

This has been proven by examples, mathematics, simulations, real cases, and many other ways, but still we tend to bite more than we can chew and open as many tasks as possible, as if it was making us more productive. Kanban systems provide a tool to control this trend and, thus, improve both capacity and lead time. For Kanban and WIP limits to work, a small batch size is needed. Small batches are, hence, one of the paradigms of the Lean production systems, allowing fast exchange of product lines, low WIP limits, lower lead times, etc.

When you look at true Kanban systems, you realize that they address the need to understand value, see the whole value stream, help the system flow, work under pull events, and improve the whole. So Kanban is possibly the simplest way to implement the five root principles of Lean with a single tool!

Again, this is not a book on Agile or Lean tools, but you will find very valuable resources on Kanban and other Lean tools at the end of this chapter.

The NUMMI Experiment: Lean Everywhere

At the beginning of the studies on the Japanese production methods, some Lean skeptics argued that this kind of productivity was only possible in Japanese environments, where the workforce was so alienated and submissive that they will work themselves to death. Well, in fact, there is a Japanese word for that: *karoshi*. . .

Some other skeptics claimed that the Japanese competitive advantage was due to a devaluated yen, while others pointed at superior automation. But all these arguments were silenced after the NUMMI experiment.

At the beginning of the 1980s, General Motors was having big trouble with the NUMMI manufacturing plant in Fremont, California. It was described by company officials as "one of the worst in the industry": the absenteeism was so high that sometimes the whole plant shut down because there were not enough people to operate it, and frequently workers came drunk to work and sabotaged the cars they were making with small pranks, like placing beer cans inside the frame so they would rattle while driving. Quality was awful, costs were outrageous, and capacity was embarrassingly low.

General Motors ended up closing the plant in 1982. But in those times, Toyota was looking for a way to better introduce their products in the United States, and General Motors was also eager to know more about the Toyota Production System. So the two companies reopened the plant in 1984 as a joint venture – with 85% of the original workforce.

You can imagine the faces of the General Motors managers when, few years later, Fremont's NUMMI plant started assembling cars in 19 h (instead of GM's average of 31), with 45 defects per 100 cars (instead of GM's average 130), operating with 2 h of inventory (instead of GM's average of 2 weeks), no space for rework (instead of GM's average of 15%), and with no absenteeism at all (instead of GM's average of 15%).

GM was shocked. And the most shocking fact was that the whole workforce was hired from the former NUMMI workers, but *virtually no former top manager was rehired*. That means that management was Japanese and workforce was American.

When Toyota Lean Senseis were asked about this fact, they simply replied "the problem with your managers is that they want to rule, command and control, not to teach." Furthermore, observers described the way Japanese managers collaborated with American workers as "comanagement."

So yes, Lean can be implemented everywhere, and yes, management can be the big difference – or the big impediment – when trying to implement and improve a Lean system. If you tend to look around and say to yourself "oh, that would never work here," try to imagine a drunken American car worker placing beer cans inside the product, and try to explain how worse is your situation right now.

Lean in a Nutshell

Summarizing Lean in a couple of pages is a lost-in-advance battle, but I'll try to give some ideas on how a Lean system feels like. . ..

A Lean system is devoted to understand how we produce value from a customer perspective and then arrange all our processes and activities in a way that creates the most value and minimizes waste. Products are developed just in time with the minimum possible amount of time, space, and costs. There is a constant effort in lowering production time as a way to increase productivity and suit best the needs of the customer.

Work flows in small batches from the beginning of the production line without stop, and bottlenecks are rapidly addressed to maintain the line flow. The work in progress is limited to a certain amount, with "single-piece flow" (meaning that you only work on one thing at a time) being the ideal state.

Work is standardized not as a way to increase bureaucracy but as a common arrangement of how things should be done. When a problem or impediment is detected, everyone in the plant is empowered to take risks, stop the line, and swarm around the problem to not only solve it now, but also make sure that this problem will never rise again. Standards of work are updated constantly to include the solutions to the detected problems, and the whole system is constantly improved.

There are neither individual competitions nor blame-avoiding games, as everyone is aligned in the aim of improving the whole. Errors are tolerated and seen as an opportunity to improve, so there is no fear to take risks or point out mistakes. Work groups self-organize to find the best way to

optimize the system by experimenting and innovating in inspect-and-adapt cycles.

Visual boards are available everywhere so the state of work and any possible issues are detected as fast as possible and seen by everyone in the plant. Managers are available for work groups, and they will help them understand the job to be done and remove any impediments for them. They will also help them raise any issue to the company level and reach their full potential.

> "The fundamental principle of successful management is to allow subordinates to make full use of their ability"
>
> –Kaoru Ishikawa's *What is Totally Quality Control? The Japanese Way*

Quality is seen not only as conformance to requirements that has to be tested at the end of the line. A holistic approach to quality is embraced, and everyone is responsible for building quality into the whole process.

Kaizen events are frequent, and everyone in the company is required to contribute to the innovation and continuous improvement of both the product and the system. Company strategy in the long term is honored on daily short-term decisions, even at the expense of short-term losses, and this long-term thinking is also applied to company development, training, improvement, and constant reflection (Hansei).

The Pre-Manifesto Years: Agile Genealogy

Unfortunately, while the car industry was changing the world of manufacturing into Lean production, the information technology or IT industry was not catching up.

For decades, the paradigm of "software engineering" rested on the foundation of perfectly defining what was to be built and then starting to build it, which seems logical on a first thought. Hence, a huge number of methods and disciplines were designed to define, estimate, and plan software development before a single line of code was written. The worse the results, the more the software industry tried to introduce even more documents, specifications, timetables, schedules, processes, milestones, etc.

And the results were bad. Standish Group, a Boston-based project management and consulting firm, started to periodically release the "Chaos Report" about the state of project management in the IT industry. The report is based on hundreds of cases, and the underlying premise is, paraphrasing 1986's president of Transarc Corporation Alfred Spector, that bridges are normally built on time, on budget, and do not fall down, but, on the other hand, software never comes in on time or on budget and, in addition, it always breaks down.[3]

In fact, the numbers repetitively shown by Chaos Reports are scary: according to the participants, in 2009, only 32% of all projects succeeded (were delivered on time, on budget, with required features and functions), while 44% were challenged (late, over budget, and/or with less than the required features and functions) and 24% failed (canceled or delivered and never used).[4] Furthermore, the Standish reports consistently show as low as 50 or 60 cents of value delivered for every dollar spent on software, meaning that half the cost of building software is what Lean Senseis call "waste."

Still, the software industry's standard response for three decades has been "we have to estimate better – if we make perfect estimations, then our projects will always be on time, on budget and will have all the required functionality; if bridge builders can estimate, why can't we?"

At the same time, a few IT projects were beating the industry's average performance by several orders of magnitude. Small teams of no more than eight people were capable of delivering thousands of lines of working software – every week! Experts started to trace and study those projects, and what they found there was against all the established lore in IT project management. These teams were self-organizing to produce working software on short iterations, obtaining frequent feedback from the client and then rapidly adapting and extending the working software to the new requirements. Quality was outstanding, and several strategies were used to automate as many parts of the process as possible, including testing, building, and deployment. A new set of practices and tools started to emerge from those teams, including pair programming, short releases, automated building and testing, test-driven development, etc.

[3] Sachs I (2011) CHAOS report. In: Performance-driven IT management: five practical steps to business success. Government Institutes.

[4] Standish Group Press release, http://www.standishgroup.com/newsroom/chaos_2009.php

So we had some people empirically experimenting the first principles and practices of Agile Software Development. But it was not the only path that guided some brilliant pioneers to the concept. Some of them started by analyzing the Lean body of knowledge and bumped into a seminal paper by Ikujiro Nonaka and Hirotaka Takeuchi: *The New New Product Development Game*.[5] In this 1986 paper, Nonaka and Takeuchi proposed that it wasn't enough anymore to have a good product at a good price to survive – it was also necessary to adapt to an ever-changing demand from customers who now required companies to evolve their products constantly, and it was also necessary to be able do it fast! Nonaka and Takeuchi found that companies following a "waterfall" methodology of development – meaning that requirements, design, building, testing, and delivery were sequential phases usually performed by different groups of people – were not achieving the best results.

Instead, cross-functional teams of people with different skill sets that were iterating and adapting requirements, design, building, and deployment at the same time were achieving the best results in terms of creativity, innovation, productivity, quality, and time to market. Nonaka and Takeuchi compared the two approaches to a relay race, where the project was the baton and was handed from one group to another at the end of each phase, and a rugby scrum, where the project was the rugby ball and the whole team pushed it further.

In the 1990s, other experts started to apply Queue Theory and Theory of Constraints to software development, dividing the project in small batches and trying to move them through the development process as fast as they could removing bottlenecks. This was of course inspired by the Lean practices of flow and pull.

Simultaneously, some scientists studying complexity science theorized that software development is a complex field, meaning that requirements are not stable: they change when the client starts to use the product and finds new needs or discards functionality that he originally thought he needed, or even change because the environment and the problem constantly change while the software is being developed, and thus, a predictive method of requirements gathering and full solution design is not adequate: an inspect-and-adapt or empirical method should be used to approach software

[5] Nonaka I, Takeuchi H (1986) The new product development game. Harvard Business Review 65(1):137–144.

development as a complex problem, and products should be built iteratively and incrementally.

A whole new set of methodologies and frameworks started to emerge. Some of the most important were eXtreme Programming (XP), Scrum, Feature-Driven Development (FDD), Crystal Methodologies, or Dynamic Systems Development Method (DSDM).

The Agile Manifesto

In February of 2001, after more than two decades of research and practice, 17 of these experts gathered together in Utah and wrote the Agile Software Development Manifesto.[6] First seen as a countermovement against documentation-driven, heavyweight software development processes, the Manifesto was the foundation for a huge movement that, today, is widely considered the better way to develop software. Even today, a decade later, the Manifesto is still considered by many as the ultimate guide and assessment to see how Agile your software development process is.

The Manifesto reads:

> "We are uncovering better ways of developing
> software by doing it and helping others do it.
> Through this work we have come to value:
>
> –Individuals and interactions over processes and tools
> –Working software over comprehensive documentation
> –Customer collaboration over contract negotiation
> –Responding to change over following a plan
>
> That is, while there is value in the items on
> the right, we value the items on the left more."
>
> –Manifesto for Agile Software Development (17 signatories)

Let me give you my own view of the Manifesto.

As you see, the Manifesto starts by acknowledging the existence of many ways of doing things, but, in the opinion of the Agilists, Agile is a better

[6] http://www.agilemanifesto.org

way. That means you can still build a great product even if you don't use Agile. For me, this first part of the Manifesto is what I like to call "the great promise of Agile," which is that no matter how bad things go, the worst thing that can happen if you go Agile is to end up having the same results you had in the beginning.

The Manifesto goes on stating that we are uncovering these better ways by doing it, meaning this is not just an academic exercise, we've been doing this stuff and eating our own dog food and helping others do it, which means that this is not something that only works for us; you can do it too!

Then, the Manifesto enumerates the four foundational values of Agile development, but please keep in mind the last phrase: we value the items on the right! Agile is not a chaotic commune of software hippies demanding to ban documents, processes, contracts, tools, and plans. In fact, too many people have seen the Agile movement as a war between good developers and bad managers, and that is absolutely not the case. You only have to read the principles to find the need of software developers and business people to collaborate daily on the project.

The four values start with "individuals and interactions over processes and tools." As said, this is not a call to ban processes and tools: software development is a knowledge-sharing game, and the best way of managing knowledge is through personal interactions. Processes and tools must be designed keeping this fact in mind and they should foster communication, interaction . . .and face-to-face communication. If they keep people out of this kind of interactions, asking to fill timesheets, knowledge-based wikis, and a whole bunch of procedural forms instead, then they should be redesigned.

The second value is "working software over comprehensive documentation." Project documentation, from an Agile-Lean perspective, is waste. More project documentation does not give more functionality or value to the customer, so we should try to have as few documentation as possible – but not less! Spending the first few months of a project writing documents is not the best way of providing value to our customer, when we could be designing prototypes, proof of concepts, or technical spikes instead and evolving them into the full product in short client-driven iterations.

The third value is "customer collaboration over contract negotiation." Of course, contracts are important. But even when you are protected by a contract, if your client finds new needs and you don't put them into the product because your contract does not require you to do so, you are not providing the most possible value to your client. If both parts accept that

project will change and they are fine with that, a better way of collaborating and building products will be reached.

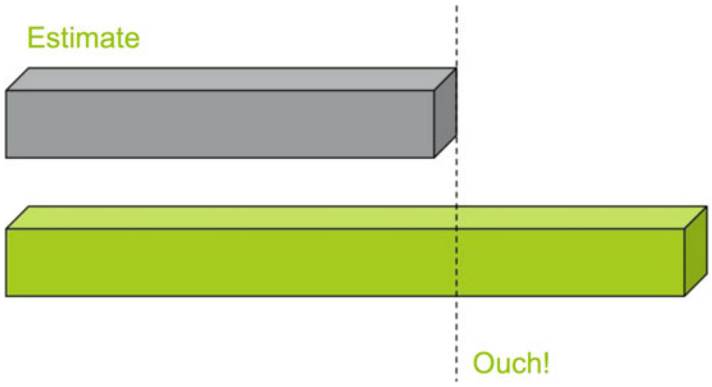
This acceptance of change is also present in the fourth value: responding to change over following a plan. Planning is important but only as far as you understand that the plan will be outdated immediately, and you have to constantly adapt to a changing environment. This is a basic principle of the universe that you have to admit to be a successful manager: you can adapt your project to reality – trying to adapt reality to your project will not work.

> "In preparing for battle I have always found that plans are useless, but planning is indispensable."
>
> –Dwight David Eisenhower, American 34th President (1953–1961)

Principles of Agile Development

The Agile Manifesto also lists 12 principles for Agile Software Development, but going in depth into each of them is out of the scope of this book. Instead, I'll give you the 10-min abstract I used to show on my seminars after six hours of Agile principles dissection.

Try to imagine traditional software development like this:

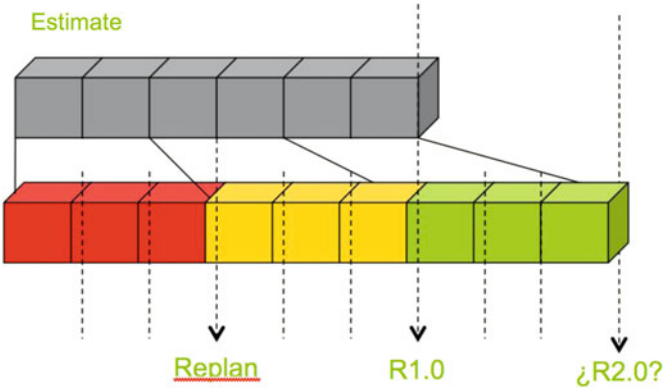


As you can see, you do some initial planning, come up with some crazy estimates, draw an aggressive deadline, and start coding like there is

no tomorrow. Some weeks before the deadline timid voices start to question it, and pessimistic reports arrive asking for more time because of unexpected events, faulty materials, wrong requirements... Oh, yes, and the client is making changes. Too many changes. So you move the deadline and prepare yourself to receive hell from both client and boss. There is nothing else to do, except urging everyone to do long hours – it is crunch time, and now you are on a death-march project!

This lowers morale and has a bad effect on product quality, which also contributes to new delays, and so not only your project is doomed: your next project will suffer from the constant interruptions, support tickets, and trouble reports that this project will cause in the future.

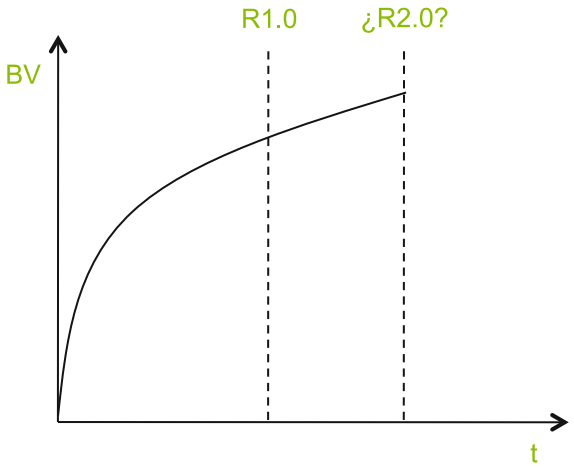
Now try to imagine a different reality:



This time you started with the same estimate you had in the other case, but now you focused on delivering the 33% of most important features (the first ones) as soon as possible. So maybe on month 3, you had a first prototype consisting of the three most important groups of features, or "epics." You realize then that they were supposed to have ended by month 2, so you clearly detect that estimates were very optimistic and it is time to replan.

What shall we do? Well, now that we are releasing working features regularly, there are more options. For instance, we could release "something" on the deadline (release 1.0) and then release the full product later (release 2.0).

But if we are delivering the most important features first, then the last features consist of the "nice to have" or "could-wish" part of the product. Maybe the cost of extending the project 3 months past the original deadline is not worth the business value we will get from these features.



This is a decision to be made by business people. In the worst case, releasing a product with only 66% of the most important features will not be acceptable to the client, but then, we would be in the first case again! So, as promised by the Manifesto, the worst thing that can happen is to finish up exactly as you were before implementing Agile. But, on the way, Agile has given you a new series of options you did not have before – like start using the 66% of the most important features before the final release.

Also, having a working product as soon as possible helps the client see what he actually needs, thus having better information than when you ask him to write on a document what he will need in the future. This has the effect of reducing risks and uncertainty, usually reducing the project time and improving the quality of the product and the satisfaction of our client.

The principles of the Manifesto describe this kind of development, where the main metric of progress is not the time elapsed, percentage of the Gantt chart done, or number of working hours put into the project, but working software delivered. Working software is the primary measure of progress, and the features we deliver are prioritized in order to satisfy our customer needs, so the most important features are delivered first. These deliveries to the customer are done on short time scales, from a couple of weeks to a

couple of months, and the delivered product should be as close to production state as possible. Every delivery creates an opportunity for our client to test the software, change the requirements, and replan the whole project, something we welcome as a way to provide competitive advantage to him.

The Agile principles establish that this kind of process must be performed by a motivated, self-organizing team of developers that is collaborating daily with business people, works at a sustainable pace, actively seeks face-to-face communication, strives for technical excellence, and frequently reflects on how to improve, simplify, maximize value, and reduce waste.

Easy, right?

Again: From Values and Principles to Practices and Tools

As we saw when studying Lean, just trying to copy the practices and tools will not take the company to a Lean state. The same happens with Agile: just trying to put some post-it notes around and conduct daily meetings will not make you Agile. Only the constant effort to embrace the values and the principles will. If you guide your company through the values and the principles in the Manifesto, it does not matter if you are doing Scrum, XP, Kanban, Lean Software Development, or the last fancy flavor of Agile. That is why, lately, many Agilists are urging companies to stop *doing* Agile and *be* Agile instead.

Anyway, the implementation of the Agile values and principles, as well as the previous work done by the Agile pioneers, has produced a well-documented set of best practices that very possibly will help you understand and embrace Agile. Some of them are described below, but, again, let me stress that even if you start performing all of them, that will not necessarily make you Agile.

From my own perspective, Agile companies must understand values and principles first and then try to find the set of processes, roles, practices, artifacts, and tools that help them live by those values and principles *and not the other way round*. There are outstanding Agile software tools out there that will help you manage your projects, collaborate with other developers, automate your testing, building, and deployment, or even measure the quality of your code. Be sure none of them will *make you Agile*. Here is some advice: If you can spend some resources on the Agile adoption process, be sure to spend it on people and not on software.

Some Agile Practices and Tools

Please be sure to read the previous section before going on. Twice. Now, you are encouraged to use as many of these practices as possible, as they have proven to be very useful and rewarding, not only to help Agile adoption but to improve team collaboration and overall performance in all kinds of projects:

- *Cross-functional teams*: Instead of setting up a team of analysts, a team of developers, and a team of testers, try to make a team of 5–9 people that includes analysts, coders, and testers. Then ask them to collaborate and produce working software together every couple of weeks, which will probably force the analysts to do some testing or the testers to do some coding. Be sure to measure and rank the team as a whole depending on the main measure of progress: working software delivered! This will introduce some problems with hyper-specialists or people whose skills are seldom used and may not have enough work to do in a team of 5–9: we will deal with this in the chapter about Agile structures.
- *Iterative and incremental development*: Instead of building block A, and then block B, and then block C, only to find out that nothing really works until block Z is in place, try to deliver a small working version of the system as soon as possible. Try to live by the philosophy of "Release early, release often and listen to your customers" proposed by Eric S. Raymond in his essay "The Cathedral and the Bazaar".[7] Several approaches have been proposed for this, including Alistair Cockburn's walking skeleton or the Minimum Marketable Feature Set/Minimum Viable Product concept.
- *Daily meetings*: Meetings are often seen as a terrible waste of time. The solution is not to have fewer meetings (well, maybe just a few less) but to be more efficient in their moderation and facilitation. On the other hand, in groups of people often called "teams," it is not rare to find individuals who don't know what their colleagues are doing in any given moment and think that they only have to worry about their personal bit of stuff. A daily five to ten minutes face-to-face meeting where every team member reports his activities, progress, and impediments is a great way of both updating the information about the project, better knowing our

[7] Raymond ES (1999) The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary. O'Reilly Media.

colleagues, sharing knowledge, showing some interest, fostering collaboration, and growing better teams.

- *Feature-Driven Development*: If we have to deliver working software every 2 weeks or so, a high stress has to be put on developing working features as a whole, instead of spending some months designing the architecture, then another few months developing the core, then application logic, then user interfaces. Evolving architecture can be a pain when you've never done it before, but feature-driven teams can react better to the feedback coming from the customers and reduce risks by, for example, not investing too much on the wrong architecture or the wrong product. Nothing is more inefficient than developing very efficiently a product that nobody wants. Feature-driven cross-functional teams will tend to be generalists, as technically specialized teams will not be able to easily change from one feature to another as demand changes. Feature-driven teams often use user stories as units of work to develop, with every user story describing a feature to be incorporated to the existing product. User stories are supposed to include a description of who wants the feature, what does he want, why (the problem to be solved), and how to test the feature at the end of the iteration. They are frequently handwritten on an index card and posted on the teams' iteration board.

- *Planning game*: In more traditional environments, a project manager plans the whole project, divides work, estimates effort and costs, and then blames everyone for not following the plan and not working hard enough. Agile teams participate as a whole in project planning, as everyone will have valuable information in his part of the project. Estimates are shared and discussed as a way to have more conversations about the product we are building. Teams that share planning are also more committed to delivery dates than those whose delivery date has been set up by someone seen as external to the team. The use of games like planning poker, story points, or workshops to create user story maps is frequent in Agile teams.

- *Co-location*: The benefits of co-locating the cross-functional team together have been widely explained in literature. The time to address the person that could have the answer to your needs drops dramatically, collaboration starts to happen naturally, and osmotic communication takes place. Alistair Cockburn, father of the concept of osmotic communication, says that "information flows into the background hearing of

members of the team, so that they pick up relevant information as though by osmosis".[8]

- *Pair programming*: This practice started among developers, but it is gradually being applied by other members of the team. For example, developers and testers are pairing, so code is easier to test. And analysts and coders are pairing so coders better understand what they need to solve, and also produce more readable code. When two people pair, only one computer is used. While one of them is at the keyboard, the other plays the navigator or observer role. Pair programming has proven to produce better code quality, hence reducing the number of bugs and improving performance. It is also a great way of learning and sharing knowledge between team members.
- *Visual management*: Agile teams love to post things on their walls. Iteration state, who is working on what, project progress, impediments, definition of "Done"... even appreciation messages from one team member to another. The Agile team's "Visual War Room" has a lot of influence on team members: it puts some peer pressure on them, while also creating a team culture of what is important for them and what they are striving for. A direct heritage from Lean environments, no software tool has ever matched the subtle power of physical team boards!
- *Agile coach*: Agile companies have found teamwork to be the ultimate competitive advantage. But they've also learned that putting some people together, throwing a project to them, and calling them "team" will not necessarily produce the kind of magic they are searching for. The role of the Agile coach is both teaching Agile practices and coaching the team to make the most out of them. This usually includes coaching individual team members, helping the team solve their conflicts, teaching them to communicate in a constructive way, facilitating meetings, making them trust one another, or showing them ways to remove impediments by themselves.

> "It is teamwork that remains the ultimate competitive advantage, both because it is so powerful and so rare."
>
> –Patrick Lencioni, *The Five Dysfunctions of a Team*

[8] Cockburn A (2004) Crystal clear: a human powered methodology for small teams. Addison-Wesley Professional.

- *Retrospectives*: At frequent intervals, the team reflects on what is going well, and how to spread those practices, and what is going wrong, so they find ways to improve it. The behavior of the team is automatically changed in order to implement the improvement plan they have designed, and the appropriate processes and documentation are updated to show these changes.

Agile also has a wide spectrum of technical practices for developers. Agile developers share the ownership of the whole application, so everyone is entitled to modify someone else's code. Tests are written prior to the code, and then only the code needed to pass the tests is written. Once all tests are passing, the code is refactored to make it simpler, smaller, and more efficient. Every piece of finished code is automatically built into the whole application, and alarms stop the development process if the build is broken – again, a direct heritage from Lean environments, where the Jidoka or autonomation principle will stop the whole production line when a problem is detected.

Agile in a Nutshell

Again, trying to summarize Agile seems a Herculean task, but I'll try to give an overall image of how an Agile environment feels like:

It's Monday morning on Agile Enterprise Inc. A new project is launching, so a User Story Mapping workshop has been set up by the Agile Coach. Representatives from the client have come to explain their problem to the team, and the team starts to make questions about the desired behavior of the system they will be building. Groups of features (epics) start to be identified and posted to the wall, and a general skeleton of the application takes form. After several hours, some hundred features have been defined at high level and prioritized by the client, so the coach calls for the workshop to end. A short retrospective of the workshop is conducted to see what can be improved next time and how valuable was the time invested.

On Tuesday the team plans for an iteration of two weeks, and they estimate that the first twelve features can be done by the end of the iteration. An iteration board is designed showing the features to be developed in "pending" state. While the iteration goes on, each feature will be moved to "analysis," "development," "testing," and "done!" to show their progress. The team will make an effort to have

(continued)

as few open features at a time as possible, thus reducing the work in progress and the context switching. This will give the team better concentration and performance, and the overall lead time will also improve.

Every day the team will gather together standing up in front of the board and tell the other team members what they are working at, how much is still to be done, what they will be doing next and what problems and impediments they are facing. The board will be updated, so it will always show the current iteration state, and the team will go back to work. The coach will take note of the impediments and help the team when they find one that blocks a feature: their priority will be always to solve the impediment, and not to abandon the feature in favor of an easier, unblocked but less valuable and less priority item.

Exactly two weeks later, not a day after, a meeting with the client takes place. The team managed to successfully finish ten out of the twelve planned features. Unexpected technical issues were discovered: fortunately enough, we are still on time to re-plan the whole project, as only two weeks have been invested and the knowledge we have right now, after physically working on the code for two weeks, is much higher than the one we had when thinking and designing with pencil and paper.

The ten features or user stories are accepted by the client, but now that he sees and uses the features and knows how they feel, he realizes that some new features will be needed. He also notices that some of the features on the original backlog will not be very valuable, as the current features cover most of the functionality that the non-developed features will provide, so some changes and re-arrangements are made to the feature backlog, and the team plans for a new iteration.

Before starting the new iteration, the Coach calls for an iteration retrospective. The team is happy with their collaboration level and the way they are reporting and managing the project, but they are a bit upset about the technical issues they weren't able to predict. All of them have something to do with an arcane module of their legacy code-base that nobody really knows how it works. So they start thinking about a way to refactor this module and turn it into something more stable, bug-free, and documented. Some capacity of the team will be spared to work on this module and, while this will drop the team's velocity for a couple of iterations, reducing the number of blocked stories and impediments will probably rise the team's throughput later.

Summary

Lean and Agile are frameworks that very often seem counterintuitive. We've usually been educated on Taylor's paradigms of man-hours, carrots and sticks, bosses and subordinates, hyper-specialization, and *Dilbertesque* office environments. While companies have fancy mission statements that state the importance of the customer, the usual fact is that they ignore and abuse them in order to raise their profit, and while the companies often say that their workers are their most important asset, downsizing, offshoring, and institutionalized abuse, harassment, and exploitation are in the order of business.

> "You are not paid to think. A mindless worker is a happy worker. Shut up and do your job!"
>
> –Futurama de-motivational poster

On the other hand, Lean and Agile rest on the foundation of real client focus – maximizing value from the client's perspective, comanagement, teamwork, empowerment, motivation, values, pride... Put it this way, it is normal that many people will say "that will never work", but it is working. It has been working for decades, in fact.

Lean companies are four times as productive as their market's average, their quality is ten times better, and their costs are less than half that of their competitors. These rates skyrocket if we study companies based on knowledge workers, where the best of their field can be 30 or 50 times better than the average, with this number growing even more when the knowledge field gets more complex.

Companies working by the old twentieth-century paradigms face a dramatic decision: if they go on trying to brute-force their business by lowering costs, they will need to fight against the rising economies like China, Korea, Brazil, India, or Russia, where labor costs are cheaper and people usually work for 12 h a day, 6 days a week.

On the other hand, they can embrace Japan's model of efficiency, quality, and continuous improvement, but this is not possible without a committed and aligned workforce.

So the question is, how can we turn our listless and individualized workforce into a Lean-Agile set of true collaborating teams capable of beating the heck out of the market?

This, above all, is the Agile manager's role.

Things to Try

- Visualize your value stream. Find some books or tutorials on the topic, and then start by mapping your whole production process backward with the help of no more than a pencil and a notebook. Start with the deliveries of the goods, then go back step by step until you reach the point where an order from the client was received. Don't trust the people's estimate: measure the time by tracing the orders, receipts, delivery tickets, e-mails, and any other available information. Try to find out how much time was spent really working on the product and how much was wasted in queues, waiting time, handoffs, reworks. . . Identify bottlenecks and design a plan to exploit them.
- List your main sources of waste. Read about Muda, Mura, and Muri as a way to inspire yourself and learn to see waste around. Look carefully at every activity and ask yourself "do we really want to do more of it?" If not, label it as waste and try to reduce it in the future. Try to quantify the cost this activity has in the overall production process. Make everyone join you in this search for waste.
- Start to schedule regular Gemba Walks: go to the place where actual value is created, the workplace, and spend some time wandering around trying to see if help is needed. Also try to physically follow the value stream from side to side and spot hidden sources of waste, bottlenecks, current issues, and possible improvements. Don't use Gemba Walks to interrupt people and ask them about the state of work: only address people if a problem in the system as a whole has been spotted, or if they ask you for your help.
- Use your value stream to start a Kanban board. Gather all ongoing and pending work and represent it on a board with all the Value Stream steps. You can find excellent recommended tutorials on Kanban at the end of this chapter. Note down every step the work takes and start measuring cycle time and lead time. Use your Kanban board to encourage discussion and reflection among teams about current issues, bottlenecks, prioritization schemas, sources of waste, and improvements of the current process.

- Start a visual management program. Organizations are as mature as the information is free to travel across them. If people want to hide information, there is something to be addressed there. Urge everyone to make the information about their projects immediately visible and constantly updated. Reinforce the visual management program with an A3 initiative: urge everyone to design A3 forms for their reports and documents.
- Launch a Kaizen program. You can start by asking everyone to do periodical retrospectives – every 2 or 3 weeks is a good place to start. Ask people to list things that are working well and things that are not. Every retrospective exercise should end with a list of proposed actions – if not, it's just whining!
- Ask your coworkers what problems they face in their daily jobs. If they answer "No problem!" make them realize that "No problem" is a problem, as it hides the path to continuous improvement. Try to ask them what they would need to double productivity with the same resources they have right now.
- For every recurrent problem or error detected, launch a root cause analysis workshop. Try to find the roots of the problem by asking, "Why?" (Toyota is well known for using five why's for every detected problem). Learn about Ishikawa fishbone diagrams, as well as cause-effect diagrams.
- Think of a way to work on smaller work batches more frequently. Try to release something to your client every 2 weeks or so, gather client's feedback, and then replan according to the feedback received. Invite your client to the planning meetings.
- Train your people on Agile and Lean. Then, let them figure out what parts of Agile and Lean they could be using. Don't try to implement the whole set at once – you'll inevitably fail! Start by understanding the core principles and values instead.
- If you work at a software development or IT company, gradually introduce some of the Agile practices described. Find more practices and tools in some of the recommended readings. Even if you are not at a software company, try to introduce some work-in-pairs time every week: you'll be amazed with the results! Remember to rotate the pairs in order to obtain the best results in terms of knowledge sharing.
- Find and join some Lean or Agile practitioners group in your city. If there is not one, join an online group. You'll find interesting communities of practice in Yahoo groups, Google groups, or LinkedIn.

Recommended Readings

Anderson DJ (2010) Kanban – successful evolutionary change for your technology business. Blue Hole Press, Sequim

Beck K (1999) Extreme programming explained: embrace change. Addison-Wesley Professional, Boston

Cockburn A (2006) Agile software development: the cooperative game. Addison-Wesley Professional, Upper Saddle River

Cohn MW (2009) Succeeding with agile: software development using scrum. Addison-Wesley Professional, Upper Saddle River

Dowser J (1999) Embracing defeat: Japan in the wake of World War II. W. W. Norton & Company, New York

Ishikawa K (1991) What is totally quality control? The Japanese way. Productivity Press, Cambridge

Kniberg H, Skarin M (2010) Kanban and Scrum – making the most of both. lulu.com, Raleigh

Liker JK (2003) The Toyota way: 14 management principles from the world's greatest manufacturer. McGraw-Hill, New York

Nonaka I, Takeuchi H (1995) The knowledge-creating company: how Japanese companies create the dynamics of innovation. Oxford University Press, New York

Ohno T (1988) Toyota production system: beyond large-scale production. Productivity Press, Portland

Ohno T (2009) Workplace management. Gemba Press, Mukilteo

Poppendieck M, Poppendieck T (2003) Lean software development: an Agile toolkit. Addison-Wesley Professional, Boston

Poppendieck M, Poppendieck T (2006) Implementing lean software development: from concept to cash. Addison-Wesley Professional, Boston

Poppendieck M, Poppendieck T (2009) Leading lean software development: results are not the point. Addison-Wesley Professional, Boston

Rother M, Shook J (1999) Learning to see: value-stream mapping to create value and eliminate Muda. Lean Enterprise Institute, Cambridge

Shingo S (2007) Kaizen and the art of creative thinking: the scientific thinking mechanism. PCS Inc. and Enna Products Corporation, Vancouver

Shingo S (2009) Fundamental principles of lean manufacturing. PCS Inc. and Enna Products Corporation, Vancouver

Tabb WK (1995) The postwar Japanese system: cultural economy and economic transformation. Oxford University Press, New York

Womack JP, Jones DT (2003) Lean thinking: Banish waste and create wealth in your corporation. Free Press, New York

Womack JP, Jones DT, Roos D (1991) The machine that changed the world: the story of lean production. Harper Perennial, New York