Principles of Lean Thinking

Mary Poppendieck
Poppendieck.LLC
7666 Carnelian Lane
Eden Prairie, MN 55346 USA
952-934-7998
mary@poppendieck.com

Abstract

In the 1980's, a massive paradigm shift hit factories throughout the US and Europe. Mass production and scientific management techniques from the early 1900's were questioned as Japanese manufacturing companies demonstrated that 'Just-in-Time' was a better paradigm. The widely adopted Japanese manufacturing concepts came to be known as 'lean production'. In time, the abstractions behind lean production spread to logistics, and from there to the military, to construction, and to the service industry. As it turns out, principles of lean thinking are universal and have been applied successfully across many disciplines.

Lean principles have proven not only to be universal, but to be universally successful at improving results. When appropriately applied, lean thinking is a well-understood and well-tested platform upon which to build agile software development practices.

Introduction

Call a doctor for a routine appointment and chances are it will be scheduled a few weeks later. But one large HMO in Minnesota schedules almost all patients within a day or two of their call, for just about any kind of medical service. A while ago, this HMO decided to worked off their schedule backlogs by extending their hours, and then vary their hours slightly from week to week to keep the backlog to about a day. True, the doctors don't have the comforting weeks-long list of scheduled patients, but in fact, they see just as many patients for the same reasons as they did before. The patients are much happier, and doctors detect medical problems far earlier than they used to.

The idea of delivering packages overnight was novel when Federal Express was started in 1971. In 1983, a new company called Lens Crafters changed the basis of competition in the eyeglasses industry by assembling prescription glasses in an hour. The concept of shipping products the same day they were ordered was a breakthrough concept when LL Bean upgraded its distribution system in the late 1980's. Southwest Airlines, one of the few profitable airlines these days, saves a lot of money with its unorthodox method of assigning seats as people arrive at the airport. Dell maintains profitability in a cutthroat market by manufacturing to order in less than a week. Another Austin company builds custom homes in 30 days.

The common denominator behind these and many other industry-rattling success stories is lean thinking. Lean thinking looks at the value chain and asks: How can things be structured so that the enterprise does nothing but add value, and does that as rapidly as possible? All the intermediate steps, all the intermediate time and all the intermediate people are eliminated. All that's left are the time, the people and the activities that add value for the customer.

Origins of Lean Thinking

Lean thinking got its name from a 1990's best seller called *The Machine That Changed the World : The Story of Lean Production*[1]. This book chronicles the movement of automobile manufacturing from craft production to mass production to lean production. It tells the story of how Henry Ford standardized automobile parts and assembly techniques, so that low skilled workers and specialized machines could make cheap cars for the masses. The book goes on to describe how mass production provided cheaper cars than the craft production, but resulted an explosion of indirect labor: production planning, engineering, and management. Then the book explains how a small company set its sights set on manufacturing cars for Japan, but it could not afford the enormous investment in single purpose machines that seemed to be required.

[1] *The Machine That Changed the World : The Story of Lean Production*, by Womack, James P., Daniel T. Jones, and Daniel Roos, New York: Rawson and Associates; 1990.

Nor could it afford the inventory or large amount of indirect labor that seemed necessary for mass production. So it invented a better way to do things, using very low inventory and moving decision-making to production workers. Now this small company has grown into a large company, and the Toyota Production System has become known as 'lean production'.

"The mass-producer uses narrowly skilled professionals to design products make by unskilled or semiskilled workers tending expensive, single-purpose machines. These churn out standardized products at high volume. Because the machinery costs so much and is so intolerant of disruption, the mass-producer adds many buffers – extra supplies, extra workers, and extra space – to assure smooth production…. The result: The customer gets lower costs but at the expense of variety and by means of work methods that most employees find boring and dispiriting."[2]

Think of the centralized eyeglasses laboratory. Remember that Sears used to take two or three weeks to fill orders from its once-popular catalog. Recall the long distribution channel that used to be standard in the computer market. Think dinosaurs. Centralized equipment, huge distribution centers and lengthy distribution channels were created to realize economies of scale. They are the side effects of mass-production, passed on to other industries. What people tend to overlook is that mass-production creates a tremendous amount of work that does not directly add value. Shipping eyeglasses to a factory for one hour of processing adds more handling time by far than the processing time to make the glasses. Adding retail distribution to the cutthroat personal computer industry means that a manufacturer needs six weeks to respond to changing technology, instead of six days. Sears' practice of building an inventory of mail orders to fill meant keeping track of stacks of orders, not to mention responding to innumerable order status queries and constant order changes.

"The lean producer, by contrast, combines the advantages of craft and mass production, while avoiding the high cost of the former and the rigidity of the later… Lean production is 'lean' because it uses less of everything compared with mass production – half the human effort in the factory, half the manufacturing space, half the investment in tools, half the engineering hours to develop a new product in half the time. Also, it requires keeping far less than half the inventory on site, results in many fewer defects, and produces a greater and ever growing variety of products."[3]

While on a tour of a large customer, Michael Dell saw technicians customizing new Dell computers with their company's 'standard' hardware and software. "Do you think you guys could do this for me?" his host asked. Without missing a beat, Dell replied, "Absolutely, we'd love to do that."[4] Within a couple of weeks, Dell was shipping computers with factory-installed, customer-specific hardware and software. What took the customer an hour could be done in the factory in minutes, and furthermore, computers could be shipped directly to end-users rather than making a stop in the corporate IT department. This shortening of the value chain is the essence of lean thinking.

Companies that re-think the value chain and find ways to provide what their customers value with significantly fewer resources than their competitors can develop an unassailable competitive advantage. Sometimes competitors are simply not able to deliver the new value proposition. (Many have tired to copy Dell; few have succeeded.) Sometimes competitors do not care to copy a new concept. (Southwest Airlines has not changed the industry's approach to seat assignments.) Sometimes the industry follows the leader, but it takes time. (Almost all direct merchandise is shipped within a day or two of receiving an order these days, but the Sears catalog has been discontinued.)

Lean Thinking in Software Development

eBay is a company which pretty much invented 'lean' trading by eliminating all the unnecessary steps in the trading value chain. In the mid 1990's, basic eBay software capabilities were developed by responding daily to customer requests for improvements.[5] Customers would send an e-mail to Pierre Omidyar with a suggestion and he would implement the idea on the site that night. The most popular features of eBay, those which create the highest competitive advantage, were created in this manner.

Digital River invented the software download market in the mid 1990's by focusing on 'lean' software delivery. Today Digital River routinely designs and deploys

[2] Womack (1990) p 13.

[3] Womack (1990) p 13.

[4] *Direct from Dell*, by Michael Dell with Catherine Fredman, Harper Business, 1999, p 159

[5] *Q&A with eBay's Pierre Omidyar*, Business Week Online, December 3, 2001.

sophisticated web sites for corporate customers in a matter of a weeks, by tying the corporation's legacy databases to standard front end components customized with a 'look and feel' specific to each customer.

In the mid 1990's, Microsoft implemented corporate-wide financial, purchasing and human resource packages linked to data warehouses which can be accessed via web front-ends. Each was implemented by "a handful of seasoned IT and functional experts… (who got) the job done in the time it takes a … committee to decide on its goals."[6]

In each of these examples, the focus of software development was on rapid response to an identified need. Mechanisms were put in place to dramatically shorten the time from problem recognition to software solution. You might call it 'Just-in-Time' software development.

The question is – why isn't all software developed quickly? The answer is – rapid development must be considered important before it becomes a reality. Once speed becomes a value, a paradigm shift has to take place, changing software development practices from the mass production paradigm to lean thinking.

If your company writes reams of requirements documents (equivalent to inventory), spends hours upon hours tracking change control (equivalent to order tracking), and has an office which defines and monitors the software development process (equivalent to industrial engineering), you are operating with mass-production paradigms. Think 'lean' and you will find a better way.

Basic Principles of Lean Development

There are four basic principles of lean thinking which are most relevant to software development:

The Basic Principles of Lean Development
Add Nothing But Value (Eliminate Waste)
Center On The People Who Add Value
Flow Value From Demand (Delay Commitment)
Optimize Across Organizations

[6] *Inside Microsoft: Balancing Creativity and Discipline*, Herbold, Robert J.; *Harvard Business Review*, January 2002.

Add Nothing But Value (Eliminate Waste)

The first step in lean thinking is to understand what value is and what activities and resources are absolutely necessary to create that value. Once this is understood, everything else is waste. Since no one wants to consider what they do as waste, the job of determining what value is and what adds value is something that needs to be done at a fairly high level. Let's say you are developing order tracking software. It seems like it would be very important for a customer to know the status of their order, so this would certainly add customer value. But actually, if the order is in house for less than 24 hours, the only order status that is necessary is to inform the customer that the order was received, and then that it has shipped, and let them know the shipping tracking number. Better yet, if the order can be fulfilled by downloading it on the Web, there really isn't any order status necessary at all.

To develop breakthroughs with lean thinking, the first step is learning to see waste. If something does not directly add value, it is waste. If there is a way to do without it, it is waste. Taiichi Ohno, the mastermind of the Toyota Production System, identified seven types of manufacturing waste:

The Seven Wastes of Manufacturing
Overproduction
Inventory
Extra Processing Steps
Motion
Defects
Waiting
Transportation

Here is how I would translate the seven wastes of manufacturing to software development:

The Seven Wastes of Software Development
Overproduction = Extra Features
Inventory = Requirements
Extra Processing Steps = Extra Steps
Motion = Finding Information
Defects = Defects Not Caught by Tests
Waiting = Waiting, Including Customers
Transportation = Handoffs

Extreme Programming (XP) is a set of practices which focuses on rapid software development. It is interesting to examine how XP works to eliminate the seven wastes of software development:

Waste in Software Development	How Extreme Programming Addresses Waste
Extra Features	Develop only for today's stories
Requirements	Story cards are detailed only for the current iteration
Extra Steps	Code directly from stories; get verbal clarification directly from customers
Finding Information	Have everyone in the same room; customer included
Defects Not Caught by Tests	Test first; both developer tests and customer tests
Waiting, Including Customers	Deliver in small increments
Handoffs	Developers work directly with customers

'Do It Right The First Time'

XP advocates developing software for the current need, and as more 'stories' (requirements) are added, the design should be 'refactored'[7] to accommodate the new stories. Is it waste to refactor software? Shouldn't developers "Do It Right the First Time?"

It is instructive to explore the origins of the slogan "Do It Right the First Time." In the 1980's it was very difficult to change a mass-production plant to lean production, because in mass production, workers were not expected to take responsibility for the quality of the product. To change this, the management structure of the plant had to change. "Workers respond only when there exists some sense of reciprocal obligation, a sense that management actually values skilled workers, … and is willing to delegate responsibility to [them]."[8] The slogan "Do It Right the First Time" encouraged workers to feel responsible for the products moving down the line, and encourage them to stop the line and troubleshoot problems when and where they occurred.

In the software industry, the same slogan "Do It Right the First Time," has been misused as an excuse to apply mass-production thinking, not lean thinking to software development. Under this slogan, ***responsibility has been taken away from the developers who add value,*** which is exactly the opposite of its intended effect. "Do It Right the First Time" has been used as an excuse to insert reams of paperwork and armies of analysts and designers between the customer and the developer. In fact, the slogan is only properly applied if it gives developers more, not less, involvement in the results of their work.

A more appropriate translation of such slogans as "Zero Defects" and "Do It Right the First Time" would be "Test First". In other words, don't code unless you understand what the code is supposed to do and have a way to determine whether the code works. A good knowledge of the domain coupled with short build cycles and automated testing constitute the proper way for software developers to "Do It Right the First Time".

Center On The People Who Add Value

Almost every organization claims it's people are important, but if they truly center on those who add value, they would be able to say:

The people doing the work are the center of
Resources
Information
Process Design Authority
Decision Making Authority
Organizational Energy

In mass-production, tasks are structured so that low skilled or unskilled workers can easily do the repetitive work, but engineers and managers are responsible for production. Workers are not allowed to modify or stop the line, because the focus is to maintain volume. One of the results of mass-production is that unskilled workers have no incentive to volunteer information about problems with the manufacturing line or ways to improve the process. Maladjusted parts get fixed at the end of the line; a poor die or improperly maintained tool is management's problem. Workers are neither trained nor encouraged to worry about such things.

"The truly lean plant has two key organizational features: *It transfers the maximum number of tasks and responsibilities to those workers actually adding value to the car on the line, and it has in place a system for*

[7] Refactoring is improving the design of software without changing functionality.

[8] Womack (1990) p 99.

detecting defects that quickly traces every problem, once discovered, to its ultimate cause."[9] Similarly in any lean enterprise, the focus is on the people who add value. In lean enterprises, traditional organizational structures give way to new team-oriented organizations which are centered on the flow of value, not on functional expertise.

The first experiment Taiichi Ohno undertook in developing lean production was to figure out a way to allow massive, single-purpose stamping machines to stamp out multiple parts. Formerly, it took skilled machinists hours, if not days, to change dies from one part to another. Therefore, mass production plants had many single purpose stamping machines in which the dies were almost never changed. Volume, space, and financing were not available in Japan to support such massive machines, so Ohno set about devising simple methods to change the stamping dies in minutes instead of hours. This would allow many parts of a car to be made on the same line with the same equipment. Since the workers had nothing else to do while the die was being changed, they also did the die changing, and in fact, the stamping room workers were involved in developing the methods of rapid die changeover.

Ohno transferred most of the work being done by engineers and managers in mass-production plants to the production workers. He grouped workers in small teams and trained the teams to do their own industrial engineering. Workers were encouraged to stop the line if anything went wrong, (a management job in mass-production). Before the line was re-started, the workers were expected to search for the root cause of the problem and resolve it. At first the line was stopped often, which would have been a disaster at a mass-production plant. But eventually the line ran with very few problems, because the assembly workers felt responsible to find, expose, and resolve problems as they occurred.

It is sometimes thought that a benefit of good software engineering is to allow low skilled programmers to produce code while a few high skilled architects and designers do the critical thinking. With this in mind, a project is often divided into requirements gathering, analysis, design, coding, testing, and so on, with decreasing skill presumably required at each step. A 'standard process' is developed for each step, so that low-skilled programmers, for example, can translate design into code simply by following the process.

This kind of thinking comes from mass-production, where skilled industrial engineers are expected to design production work for unskilled laborers. It is the antithesis of lean thinking and devalues the skills of the developers who actually write the code as surely as industrial engineers telling laborers how to do their jobs devalues the skills of production workers.

Centering on the people who add value means upgrading the skills of developers through training and apprenticeships. It means forming teams that design their own processes and address complete problems. It means that staff groups and managers exist to support developers, not to tell them what to do.

Flow Value From Demand (Delay Commitment)

The idea of flow is fundamental to lean production. If you do nothing but add value, then you should add the value in as rapid a flow as possible. If this is not the case, then waste builds up in the form of inventory or transportation or extra steps or wasted motion. The idea that flow should be 'pulled' from demand is also fundamental to lean production. 'Pull' means that nothing is done unless and until a downstream process requires it. The effect of 'pull' is that production is not based on forecast; commitment is delayed until demand is present to indicate what the customer really wants.

Pulling from demand can be one of the easiest ways to implement lean principles, as LL Bean and Lens Crafters and Dell found out. The idea is to fill each customer order immediately. In mass-production days, filling orders immediately meant building up lots of inventory in anticipation of customer orders. Lean production changes that. The idea is to be able to make the product so fast that it can be made to order. True, Dell and Lens Crafters and LL Bean and Toyota have to have some inventory of sub-assemblies waiting to be turned into a finished product at a moments notice. But it's amazing how little inventory is necessary, if the process to replenish the inventory is also lean. A truly lean distribution channel only works with a really lean supply chain coupled to very lean manufacturing.

The "batch and queue" habit is very hard to break. It seems counterintuitive that doing a little bit at a time at the last possible moment will give faster, better, cheaper results. But anyone designing a control system knows that a short feedback loop is far more effective at maintaining control of a process than a long loop. The problem with batches and queues is that they hide problems. The idea of lean production is to expose

[9] Womack (1990) p 99. Italics in the original.

problems as soon as they arise, so they can be corrected immediately. It may seem that lean systems are fragile, because they have no padding. But in fact, lean systems are quite robust, because they don't hide unknown, lurking problems and they don't pretend they can forecast the future.

In Lean Software Development, the idea is to maximize the flow of information and delivered value. As in lean production, maximizing flow does not mean automation. Instead, it means limiting what has to be transferred, and transferring that as few times as possible over the shortest distance with the widest communication bandwidth as late as is possible. Handing off reams of frozen documentation from one function to the next is a mass-production mentality. In Lean Software Development, the idea is to eliminate as many documents and handoffs as possible. Documents which are not useful to the customer are replaced with automated tests. These tests assure that customer value is delivered both initially and in the future when the inevitable changes are needed.

In addition to rapid, Just-in-Time information flow, Lean Software Development means rapid, Just-in-Time delivery of value. In manufacturing, the key to achieving rapid delivery is to manufacture in small batches pulled by a customer order. Similarly in software development, the key to rapid delivery is to divide the problem into small batches (increments) pulled by a customer story and customer test. The single most effective mechanism for implementing lean production is adopting Just-in-Time, pull-from-demand flow. ***Similarly, the single most effective mechanism for implementing Lean Development is delivering increments of real business value in short time-boxes.***

In Lean Software Development, the goal is to eliminate as many documents and handoffs as possible. The emphasis is to pair a skilled development team with a skilled customer team and give them the responsibility and authority to develop the system in small, rapid increments, driven by customer priority and feedback.

Optimize across Organizations

Quite often, the biggest barrier to adopting lean practices is organizational. As products move from one department to another, a big gap often develops, especially if each department has its own set of performance measurements that are unrelated to the performance measurements of neighboring departments.

For example, let's say that the ultimate performance measurement of a stamping room is machine productivity. This measurement motivates the stamping room to build up mounds of inventory to keep the machines running at top productivity. It does not matter that the inventory has been shown to degrade the overall performance of the organization. As long as the stamping room is measured primarily on machine productivity, it will build inventory. This is what is known as a sub-optimizing measurement, because it creates behavior which creates local optimization at the expense of overall optimization.

Sub-optimizing measurements are very common, and overall optimization is virtually impossible when they are in place. One of the biggest sub-optimizing measurements in software development occurs when project managers are measured on earned value. Earned value is the cost initially estimated for the tasks which have been completed. The idea is that you had better not have spent any more than you estimated. The problem is, this requires a project manager to build up an inventory of task descriptions and estimates. Just as excess inventory in the stamping room slows down production and degrades over time, the inventory of tasks required for earned value calculations gets in the way of delivering true business value and also degrades over time. Nevertheless, if there is an earned value measurement in place, project tasks are specified and estimated, and earned value is measured. When it comes to a choice between delivering business value or earned value (and it often does), earned value usually wins out.

To avoid these problems, lean organizations are usually structured around teams that maintain responsibility for overall business value, rather than intermediate measurements such as their ability to speculate and pad estimates. Another approach is to foster a keen awareness that the downstream department is a customer, and satisfying this internal customer is the ultimate performance measurement.

The paradigm shift that is required with lean thinking is often hindered if the organization is not structured around the flow of value and focused on helping the customer pull value from the enterprise. For this reason, software development teams are best structured around delivering increments of business value, with all the necessary skills on the same team (eg. customer understanding / domain knowledge, architecture / design, system development, database administration, testing, system administration, etc.).

Software Development Contracts

Flow along the value stream is particularly difficult when multiple companies are involved. Many times I have heard the lament: "Everything you say makes sense, but it is impossible to implement in our environment, because we work under contracts with other organizations." Indeed, the typical software development contract can be the ultimate sub-optimizing mechanism. Standard software contracts and supplier management practices have a tendency to interfere with many lean principles.

Manufacturing organizations used to have the same problem. For example, US automotive companies once believed the best way to reduce the cost of parts in an automobile was with annual competitive bidding. If the only thing that is important is cheap parts, competitive bidding may seem like the best way to achieve this goal. However, if overall company performance is more important, then better parts which integrate more effectively with the overall vehicle are more valuable. In fact, there is an direct correlation between an automotive company's profitability and its degree of collaboration with suppliers.[10] When Chrysler moved from opportunistic to collaborative relationships with its suppliers in the late 1990's, it's performance improved significantly.

The software industry has some lessons to learn in the area of contractual agreements between organizations. It needs to learn how to structure collaborative relationships which maximize the overall results of both parties. A key lesson the software industry needs to learn is how to structure contracts for incremental deliveries that are not pre-defined in the contract, yet assure the customer of prompt delivery of business value appropriate to their investment. Here again, we can learn from lean production.

Lean manufacturing organizations develop a limited number of relationships with 'trusted' suppliers, and in turn, gain the 'trust' of these suppliers. What does 'trust' mean? "Trust [is] one party's confidence that the other party in the exchange relationship will fulfill its promises and commitments and will not exploit its vulnerabilities."[11] "…trust…[is] not based on greater interpersonal trust, but rather greater trust in the fairness, stability, and predictability of [the company's] routines and processes."[12]

It has been the practice of legal departments writing software contracts to put into contractual language all of the protections necessary to keep the other side 'honest.' However, the transaction costs associated with creating and monitoring such contracts are enormous. Many contracts all but demand a waterfall process, even if both companies believe this is not the best approach. It's time that the software development industry learned the lesson of Supply Chain Management – "Extraordinary productivity gains in the production network or *value chain* are possible when companies are willing to collaborate in unique ways, often achieving competitive advantage by sharing resources, knowledge, and assets…. Today competition occurs between value chains and not simply between companies."[13]

Summary and Conclusion

The lean production metaphor is a good one for software development, if it is applied in keeping with the underlying spirit of lean thinking. In the past, the application of some manufacturing concepts to software development ('Do It Right the First Time' comes to mind) may have lacked a deep understanding of what makes lean principles work. The underlying principles of eliminating waste, empowering front line workers, responding immediately to customer requests, and optimizing across the value chain are fundamental to lean thinking. When applied to software development, these concepts provide a broad framework for improving software development.

[10] *Collaborative Advantage*, by Jeffrey H. Dyer, Oxford University Press; 2000, p 6.

[11] Dyer (2000) p 88.

[12] Dyer (2000) p 100

[13] Dyer (2000) p 5