Lean Programming

Mary Poppendieck, Mayo 2001
Enlace original: http://www.leanessays.com/2010/11/lean-programming.html

About the time of the 1980 NBC documentary 'If Japan Can, Why Can't We?', I was the System Manager in a video cassette manufacturing plant, and our management team was asking this question every day. Our Japanese competition was selling superior products at much lower prices, and we couldn't figure out how they did it. We knew we needed to make dramatic changes or close up shop, but we didn't know what to change.

[…]

Lean Manufacturing

At the end of World War II, Sakichi Toyoda, founder of Toyoda Spinning and Weaving company, dreamed of providing cars for the general public, much like Henry Ford's dream thirty years earlier. He chartered Taiichi Ohno to put in place an efficient production system to produce high quality automobiles. Over the next three decades, Ohno developed the Toyota Production System, now known world-wide as Lean Manufacturing[1]. The foundation Ohno's system was the absolute elimination of waste.

Ohno studied US manufacturing techniques, and learned a lot from Henry Ford's pioneer work in assembly line flow. However, the assembly line produced large lots of identical cars. Ohno didn't have the customer base to imitate the US practice of manufacturing in 'economic' (ie. large) lot sizes. He was captivated by US supermarkets, however, where a small quantity of every product was placed on shelves, and as shoppers removed products, the shelves were rapidly replenished. He decided to place inventory 'supermarkets' throughout his plant, and found that this technique dramatically lowered the 'waste' of in-process inventory. He named these inventory supermarkets 'kanban'.

Because Ohno was converting a spinning and weaving company to an automobile manufacturer, he already knew how to avoid making bad product. Founder Toyoda Sakichi had invented an automatic shut-off mechanism that stopped a weaving machine the minute a flaw such as a broken thread was detected. Ohno moved this concept to car manufacturing, where he insisted that each part be examined immediately after it was processed, and the line stopped immediately if a defect was found.

To maximize product flow, standard work sheets were created, but these were not developed at a desk by engineers. They were developed on the shop floor by the workers who know the process. Standard cycle times and kanban shelf space for each item was determined and workflow was leveled. Production workers were like a relay team, handing off the baton (product) to the next person. The handoff required 100% quality and tight timing. If things got delayed, teammates were expected to help each other set up a machine or recover from a malfunction.

Ohno's aggressive elimination of waste led him to the twin values of rapid product flow and built-in quality. Over time, Ohno discovered that these two values led to the highest quality, lowest cost, shortest lead time products possible.

Total Quality Management

About the same time, Dr. W. Edwards Deming was teaching Quality Management in Japan. In fact, the Total Quality Management (TQM) movement cannot be separated from Lean

Manufacturing. Demming's photo is in the lobby of Toyota's headquarters, bigger than the photo of founder Toyoda Sakichi. Demming didn't find an audience in the US after WW II, because managers at the time thought that poor quality was caused by people who just didn't want to do a good job. They didn't think there was much managers could do to improve quality except exhort employees to do a better job.

Demming's basic message was that quality is a management responsibility, and poor quality was almost always the result of systems imposed on workers which thwarted people's desire to do high quality work. He taught the Japanese managers how to empower production workers to investigate problems and systematically improve processes. He taught that teamwork and long term, trust-based relationships with suppliers were far better than adversarial relationships. He emphasized a culture of continuous improvement of both processes and products.

In the 1980's, Demming's fourteen points (See Appendix 1) were studied by virtually every manufacturing manager. Among these fourteen points are the well known mantra's:

- Don't Inspect Quality In.
- Constantly Improve the System.
- Break Down Barriers Between Departments.

But a few of Demming's fourteen points might seem revolutionary even today, such as:

- Drive Out Fear.
- Eliminate Quotas, Numerical Goals and Merit Ratings.
- Don't Award Business Based on Price; Minimize Total Cost.

Paradigm Shift

[…]

The critical step in implementing Lean Manufacturing in our plant was a carefully planned changeover from push scheduling to pull scheduling. We decided that we could not do it part way, we had to switch plant-wide, cold turkey, over a weekend. We devised a simple simulation which we taught to every one in the plant – managers, shift supervisors, and operators. Using the simulation, teams of workers designed the layout and flow in their areas, including the kanban cards and rapid changeover methods. The entire plant held its collective breath as the pull system went into effect, but the workers knew what to do – they had developed the methods themselves. The first week packout accuracy was 92%, and it got better from there. We were able to fill special orders in two weeks, so the vice president could stop expediting orders. In a short time we were down to one week of inventory and could fill any order in the same amount of time. We had lots of extra space, and quality had never been better.

The most difficult part of implementing Lean Manufacturing was the paradigm shift it required. Everyone 'knew' that large lot sizes were necessary to keep expensive machines running at full capacity.   They also 'knew' that machine changeovers took a long time, and every minute a machine was idle its burden rate went up. In addition, large warehouse inventories were necessary to make sure that when a customer ordered a product it could be shipped immediately. After all, customers didn't want to wait the month it took us to produce the product.

One of the reasons why Lean Manufacturing has been so difficult to implement is because people must question established, known truths, and this is not easy. Another reason is that practices which create local optimization at the expense of the overall system are difficult to recognize, let alone change. Local optimization points provide attractive points of measurement, and inevitably, what is measured is optimized.

Simple Rules

[…]

The basic practices of Lean Manufacturing and TQM in the 1980's might be summed up in these ten simple rules:

1. Eliminate Waste
2. Minimize Inventory
3. Maximize Flow
4. Pull From Demand
5. Empower Workers
6. Meet Customer Requirements
7. Do it Right the First Time
8. Abolish Local Optimization
9. Partner With Suppliers
10. Create a Culture of Continuous Improvement

These Lean Manufacturing rules have been tested and proven over the last two decades. They have been adapted to logistics, customer service, health care, finance, and even construction. The application of the rules may change slightly from one industry to the next, but the underlying principles have stood the test of time in many sectors of the economy.

Lean Programming

Recent work in Agile Methodologies, Adaptive Software Development, and Extreme Programming have in effect applied the simple rules of Lean Manufacturing to software development. The results, which we call Lean Programming, are as dramatic as the improvements in manufacturing brought on by the Just-in-Time and Total Quality Management movements of the 1980's.

Lean Rule #1: Eliminate Waste

The first rule of Lean Programming is: Eliminate waste. That is, eliminate anything which does not add value to the final product. In Lean Manufacturing, waste is identified through a value stream analysis, a process which identifies all activities in the value stream and identifies the specific value they add to the final product. The value analysis process then attempts to find a different, more efficient way to add the same value,

The documents, diagrams, and models produced as part of a software development project are often consumables, aids used to produce the system, but not necessarily a part of the final product. Once a working system is delivered, the user may care little about the intermediate consumables. Lean principles suggest that every consumable is a candidate for scrutiny. The burden is on the artifact to prove not only that it adds value to the final product, but also that it is the most efficient way of achieving that value.

Lean Rule #2: Minimize Inventory (Minimize Intermediate Artifacts)

In our manufacturing plant, we communicated this message: Inventory is waste. Why? Inventory consumes resources. Inventory slows down response time. Inventory hides quality problems. Inventory gets lost. Inventory degrades and becomes obsolete. The 'benefits' of inventory are oversold. The 'cost' of inventory almost always outweighs such 'benefits'.

The inventory of software development is documentation that is not a part of the final program. As inventory, this documentation should be subject to value analysis. Take requirements and design documents, for example. How much value do the really add? How important are they to the final product? If you compare requirements and design documents to in-process inventory, then it is striking to note that the time it takes to produce these documents probably determines the cycle time of the project. Just as inventory must be minimized to maximize manufacturing flow, so too requirements and design documents must be kept to a minimum to maximize development flow.

There are many wastes associated with this excess documentation: The waste of time producing the documents, waste of time reviewing the documents, and the work that goes into change requests and associated evaluations, priority setting, and system changes. But the biggest waste of all is the waste of building the wrong system if the documentation does not correctly and completely capture the user requirements.

The best approach for minimizing intermediate artifacts is to raise the level of abstraction of documentation. Instead of a 100 page detailed specification, write a 10 page set of rules and guidelines, and document only the exceptions. Instead of a few inches of specifics, produce a concise 25 page matrix which summarizes the effort.

We know that users are relatively poor at envisioning the details of a system from most documents, and are even less likely to correctly perceive how it should operate in their environment until they actually use it. Even if users could predict exactly how the system should operate at the present time, it is unlikely that the way the system is supposed to work months before it is delivered will be exactly the way users need it to work for the rest of its useful life. All of this must be taken into account when we determine how much value these documents actually add to the final product.

Lean Rule #3: Maximize Flow (Drive Down Development Time)

During the 1980's we learned how to make products in hours which used to take days or weeks. We learned that very rapid product flow resulted in very short cycle times, often one or two orders of magnitude lower than before. During the 1990's, e-commerce projects were often able to accomplish in weeks what used to take months or years in the traditional software development world. Yes, in some sense they cheated. But the bottom line is, huge amounts of useful software was deployed in the last five years with extremely short cycle times by traditional standards.

In a recent paper titled 'Reducing Cycle Time,'[3] Dennis Frailey proposes reducing software development cycle time using the same techniques employed to reduce manufacturing cycle time. He suggests looking for and reducing accumulations of WIP (Work in Process). Just as in manufacturing, if WIP is reduced, and the cycle time will be reduced. To reduce WIP, Frailey recommends using the 'Small Batch' principle and the 'Smooth Flow Principle', concepts straight from Lean Manufacturing.

Iterative development is basically the application of these principles to programming. The basic premise of iterative development is that small but complete portions of a system are designed and delivered throughout the development cycle, with each iteration adding an additional set of features. The cycle time from start to finish of any iteration varies from a couple of weeks to a couple of months, and each iteration engages the entire development process from gathering requirements to acceptance testing.

Lean Rule #4: Pull from Demand (Decide as Late as Possible)

In our video cassette manufacturing plant, we used to think that it would be ideal if our marketing department could forecast exact market requirements. A lot of work went into sophisticated forecasting techniques to more accurately predict the future. Then one day we realized that we were trying to do the wrong thing. It would not be ideal if we had a perfect forecast. Instead, it would be ideal if we could reduce our reliance on forecasts by reducing the system response time so dramatically that the system could respond to change rather than predict it.

In a market where volatile technology requires constant product upgrades, Dell Computer has a huge advantage over it's keenest competitors because it doesn't forecast demand, it responds to it by making-to-order in an average of six days. While Dell holds about six days of inventory, it's competitors maintain six weeks of inventory. Dell's ability to make decisions as late as possible gives Dell a significant competitive advantage in a fast-moving market.

Software development practices which keep requirements flexible as close to system delivery as possible can provide a significant competitive advantage in a changing market. In a volatile business environment, users are not able forecast their future needs accurately. Freezing the design early in a software development project is just as speculative as forecasting. Software systems should be designed to respond to change, not predict it. In software development, as in building computers, the ability to make decisions as late as possible provides a competitive advantage.

Lean Rule #5: Empower Workers (Decide as Low as Possible)

A basic principle of Lean Manufacturing is to drive decisions down to the lowest possible level, providing both the tools and the authority for people "on the floor" to make decisions. When Toyota took over GM's manufacturing plant in Fremont, California in 1983, it inherited workers with the worst productivity and absenteeism record in the industry. Those same workers doubled their quality and productivity record in two years. This was accomplished through formation of teams that were trained in work measurement and improvement techniques and expected to develop and continually improve their own work standards and practices.[4]

One of the problems with heavyweight intermediate documentation is that it attempts to make all of the decisions for developers, rather than giving them a set of guidelines. In general, raising the level of abstraction of intermediate artifacts will give guidance as well as freedom to the developers as they make the detailed design and programming decisions. It is always better to tell developers what needs to be done, not how to do it.

Developers need to understand the goal of their work and how it fits into the overall flow, what it means to meet customer requirements, and the architectural structure and GUI standards of the system. They also need to know what they must accomplish, by when, and how to tell when it is complete. Finally, their work needs to be made visible in short iterative cycles to provide the feedback necessary for continual improvement.

Lean Rule #6: Meet Customer Requirements (Now and in the Future)

In his 1979 book 'Quality is Free', Philip Crosby defines quality as 'conformance to requirements'. The Standish Group study of 1994[5] noted that the most common cause of failed projects was missing, incomplete, or incorrect requirements. The software development world has responded to this risk by amplifying the practice of gathering detailed user requirements and getting user sign-off prior to proceeding with system design. However, this approach to defining user requirements is deeply flawed.

I worked on one project in which the customer wanted a complex system delivered in ten months. Time was of the essence – 10 months or bust. And yet, being a government agency, the contract required sign-off on an external design document before internal design and coding could begin. Several users were involved, and they dragged their feet on signing the documents. Why? They were concerned that they might approve something which would prove to be a mistake later on. Since there was no easy way to change things after the design documents were signed, they took two months to approve the design. An who can blame them? Their jobs depended on them getting it right. So half way into a very tight schedule, over two months of time and a lot of paper was wasted producing and getting user sign-off on design documents.

Instead of encouraging user involvement, user sign-off tends to create an adversarial relationship between the developers and the users. Users are required to make decisions early in the development process, and are not allowed to change their minds, even when they do not have a full concept of how the system will work or how their business situation may develop in the future. Users are understandably reluctant to make these commitments, and they will instinctively delay decisions to as late in the process as possible. Note that this instinct on the part of the users is in line with Lean Rule #4.

The most effective way to accurately capture user requirements is found in the iterative approach to system development. By developing core features early and obtaining customer feedback in a focus-group demonstration of each iteration, a far more correct definition of customer requirements can be obtained. In addition, if we accept that the requirements will necessarily change over time, we must start with the essential requirement that the system must be designed to easily adapt to changes over its lifecycle.

Lean Rule #7: Do it Right The First Time (Incorporate Feedback)

Before Lean Manufacturing arrived at our plant in the early 1980's, we occasionally had output of marginal quality. We would test to find the good product and rework the bad product. After understanding the "Do it Right the First Time" rule, we closed down rework stations and stopped trying to test quality into the product. Instead, we assured that each component was good at every handoff. This involved having tests and controls at every point of manufacture to detect a drift toward out-of-spec product and stop production before any bad product was made.

"Do It Right the First Time" did not mean "Freeze the Spec". On the contrary, product specs changed constantly, and lean discipline meant being able to flawlessly adapt to changing market conditions. This was accomplished through a product architecture which facilitated manufacturing change, monitoring techniques which detected errors before they happened, and tests which were designed before manufacturing began.

In 1987 Barry Boehm observed that it costs 100 times more to find and fix a problem after software delivery than to find and fix it in early design phases.[6] This observation and the "Do it Right the First Time" rule have been widely used to justify the overhead of developing a detailed system design before code is written.

The problem lies in the assumption that it is possible to generate a detailed set of documents that correctly define customer requirements, and that those requirements will not change. The fact is that requirements do change, and frequently, over the life of most systems. "Do it Right" has also been misinterpreted to mean "don't allow changes." In fact, once we acknowledge that change is a fundamental customer requirement, it becomes clear that what "Do it Right" requires that we provide for change.

If we want to meet customer requirements, and we acknowledge that customers don't really know what they want at the beginning of development, then we need to incorporate a method of obtaining customer feedback during development. Instead, most software development practices include a "Change Control Process" which makes it so difficult to respond to user feedback that developers are discouraged from asking for it. Far from insuring a quality result, these change-resistant practices actually get in the way of "Doing it Right".

Lean Programming employs two key techniques that make change easy. Just as Lean Manufacturing builds tests into process so as to detect when the process is broken, Lean Programming builds tests into the development process in order to ensure that when changes don't inadvertently break the code. In fact, the best approach is to write the tests first, and then write the code. An excellent unit and regression testing capability is the best way to encourage change late in the development process.

The second technique for allowing change to happen late in development is refactoring, or improving the design of existing software in a controlled and rapid manner. When refactoring is an accepted practice, early designs can focus on the issue at hand rather than speculate as to what additional design elements will be needed. As the additional features are actually added, refactoring provides a new, simplified design to handle the new reality. When refactoring is a part of the process, we reduce speculation as to what will be needed in the future by making it easy to accommodate the future if and when it becomes the present.

Lean Rule #8: Abolish Local Optimization (Sub-Optimized Measurements are the Enemy)

In the 1980s, the biggest enemy of Lean Manufacturing was often the accounting department. We had big, expensive machines in our plant, and the idea that they should not be run at full capacity was radical, to put it mildly. We compiled daily reports of work-in-process inventory, and the accountants didn't want these reports abandoned just because there was virtually no WIP to report.

A generation of accountants had to retire before it was "OK" to run machines below their full capacity. Designing machines for rapid changeover rather than highest throughput remains a tough sell even today. After 20 years, Lean Manufacturing is still counter-intuitive to those who lack a broad view of the enterprise.

In this context, let's examine the role of managing scope in a software development project. Project managers have been trained to focus on managing scope, just as we in manufacturing were trained to focus on maximizing machine productivity. However, Lean Programming is fundamentally driven by time and feedback. In the same way that localized productivity optimization creates a sub-optimized overall process, so too, does focus on managing scope create a sub-optimized overall project management process.

Think about it – holding the scope to exactly what was envisioned at the beginning of a project has little value for the user whose world is changing. In fact, it adds anxiety and paralyzes decision-making. It doesn't add much to the ultimate system, which will be outdated by the time it is delivered. Managing to a scope that is no longer valid wastes a lot of time and takes up a lot of space on issue lists, trade-off negotiations and ultimate fixes to the system to get things right in the end. However, as long as keeping a project within its original scope is a key project management goal, this measurement will continue to be optimized—at the expense of the overall value delivered by the project.

Scope will take care of itself if the domain is well understood and there is a well-crafted, high-level agreement on what the system will do in the domain. Scope will take care of itself if the project is driven in time buckets that are not allowed to slip. Scope will take care of itself if both parties focus on rapid development and solving the user's problem, and adopt waste-free methods of achieving these goals.

Lean Rule #9: Partner With Suppliers (Use Evolutionary Procurement)

Lean Manufacturing did not remain in the manufacturing plant. Once the idea of partnering with suppliers was combined with an understanding of the value of rapid product flow, Supply Chain Management was born. People began to realize that it took tons of paperwork to move material between companies, and this did not add value to the product. Moreover, the paperwork cost more than one might expect, not to mention the delay in product flow that it caused. Even today, predictions of billions of dollars of savings resulting from business-to-business Web portals are based on cutting the cost of transactions required to move goods between companies.

Supply Chain Management caused companies to take a close look at their business-to-business contracts. All too often, these contracts were focused on keeping the companies from cheating each other. In addition, it was common to pit one vendor against another to assure supply and obtain the lowest cost. Again, Lean Manufacturing changed this paradigm. Deming taught that trusting relationships with single suppliers create an environment that allows optimizing the overall value to both companies.

Throughout the 1980s, companies achieved the highest quality and lowest costs in their supply chains by reducing the number of suppliers and working with the remaining suppliers as partners. The quality and creativity which resulted from collaborating supply chains was demonstrated to far outweigh the apparent (and sub-optimal) benefits that came from competitive bids and rapid turnover of suppliers. Partnering companies helped each other improve product designs and product flows. They linked systems to allow just-in-time movement of goods across several suppliers with little or no paperwork. The long-term advantages of a collaborative supply chain relationships are well documented.

Wise companies realize that traditional software development contract practices generate hidden wastes. As manufacturers discovered in the 1980s, trusted relationships with a limited set of suppliers can yield dramatic advantages. Without the adversarial relationship created by a constant focus on controlling scope and cost, software development vendors can focus on providing the best possible software for customers, fixing requirements as late as possible in the development process and providing the most value for the available money.

Lean Rule #10: Create a Culture of Continuous Improvement

When software development seems to be out of control, one response has been to increase the level of “software maturity” of the organization. This might seem to be in line with good manufacturing practice, where ISO 9000 certification and Malcom Baldridge awards are sometimes equated with excellence. However, these process documentation programs indicate excellence only when the documented process is excellent in the context of it’s use.

In many current software development projects, excellence means the ability to adapt to fast moving, rapidly changing environments. Process-intensive approaches such as the higher levels of Software Engineering Institute's (SEI) Capability Maturity Model (CMM) may lack the flexibility to respond rapidly to change. In a recent e-mail advisor from Cutter Consortium, Jim Highsmith highlights the tension between such heavyweight methodologies and lightweight methodologies such as Lean Programming.[7]

The question becomes, do process documentation certification programs stifle, rather than foster, a culture of continuous improvement? Deming would probably turn over in his grave at the thought of tomes of written processes substituting for his simple Plan-Do-Check-Act approach:

- Plan: Choose a problem. Analyze it to find a probable cause.
- Do: Run an experiment to investigate the probable cause.
- Check: Analyze the data from the experiment to validate the cause.
- Act: Refine and standardize based on the results.

Iterative development allows the use of the Plan-Do-Check-Act approach within a project. During the first iteration, the hand-off from design to programming or programming to testing may be a bit rough. It’s okay if the first iteration provides a learning experience for the project team, because there are more iterations to come, so the team can improve its process. In a sense, an iterative project environment becomes an operational environment, because processes are repeated and Deming's techniques of process improvement can be applied from one iteration to the next.

Product improvement is also possible with iterations, particularly if refactoring is used. In fact, refactoring provides a tremendous vehicle to apply the principle of continuous improvement to the programming environment.

However, we need improvements that span more than a single project. We must improve future project performance by learning from existing ones. Here again, Lean Manufacturing can point the way. During the 1980s, a set of practices summarized in the ten rules of Lean Manufacturing were adopted widely across most manufacturing plants in the West. These practices then spread to service organizations, to logistics organizations, to supply chains, and beyond. They have withstood the test of time across multiple domains.

Following the simple rules of Lean Manufacturing has brought dramatic improvements to every industry in which they have been applied. These same rules can and should be applied to software development projects. The resulting Lean Programming practices will lead to the highest quality, lowest cost, shortest lead time software development possible.

Appendix 1: Summary of W. Edwards Demming's 14 points

1. Create consistency of purpose.
2. Adopt a win-win philosophy.
3. Don't depend on mass inspection; build quality in.
4. Don't award business based on price; minimize total cost; build long-term relationships of loyalty and trust with a single suppliers.
5. Constantly improve the system of production, service, planning, etc.
6. Train for skills.
7. Provide leadership: help people do a better job.
8. Drive out fear and build trust so everyone can do a better job.
9. Break down barriers between departments; abolish competition and build a win-win system of cooperation.
10. Eliminate slogans, exhortations and zero defect targets; the cause of the bulk of problems lie in the system, and are beyond the power of workers to correct.
11. Eliminate quotas, numerical goals and Management by Objectives; substitute leadership.
12. Remove barriers that rob people of joy in their work; abolish the annual rating or merit system.
13. Educate and improve individuals.
14. Involve the entire organization.

There are many summaries of Demming's 14 points, which he modified throughout the years, in the spirit of continuous improvement. The above summary is based on his last version of the 14 points.[8].

References

[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

[2] Strategy as Simple Rules, by Eisenhardt, Kathleen M and Donald N. Sull, Harvard Business Review, Volume 79, Number 1, January 2001, pp 107- 116

[3] Reducing Cycle Time, by Frailey, Dennis, Software Development Magazine, August, 2000

[4] Time-and-Motion Regained, by Paul Adler, Harvard Business Review, January-February 1993 pp 97-108

[5]Charting the Seas of Information Technology – Chaos, by The Standish Group International, 1994

[6] Industrial Software Metrics Top 10 List, by Boehm, Barry, IEEE Software, Volume 4 Number 5, September, 1987, pp 84-85

[7] E-Projects in India, by Jim Highsmith, e-Project Advisor, Cutter Consortium's e-Project Management Advisory Service, March 1, 2001.

[8] Gone But Never Forgotten, by Brad Stratton, editor, Quality Progress Magazine, March 1994

Annotated Bibliography (Chronological)

Quality Control Handbook, Joseph M. Juran, originally published in 1951, now in its forth edition
Considered the standard reference in the field of quality. Like Demming, Juran consulted mainly in Japan during the 1950's and 60's.

Managerial Breakthrough, Joseph M. Juran, originally published in 1964
Widely ignored when it was first published, this book is now considered a landmark treatise on continuous improvement.

Quality is Free, by Philip Crosby, New York: McGraw-Hill, Inc. 1979
This is the book we used to launch the TQM program in our plant. Demming never liked the zero defects approach advocated in this book, and it contains little about statistical quality control. However, many corporate executives got the quality message from Phil Corsby.
Study of 'Toyota' Production System from Industrial Engineering Viewpoint, by Shigeo Shingo, Osaka, Japan, Shinsei Printing Co. Ltd. 1981
The title of this book is an indication of the quality of the translation, but that was not a barrier to our plant. We studied the book cover-to-cover and got our introduction to Lean Manufacturing from this book.

Toyota Production System, Practical Approach to Production Management, by Yasuhiro Monden, Norcross, Georgia, Industrial Engineering and Management Press. 1983
This is the classic book on Lean Manufacturing.

Zero Inventories, by Robert W. Hall, Homewood, IL, Dow Jones-Irwin 1983.
Robert Hall, a professor at Indiana University, understood the concept of Just-in-Time earlier than most academics. This book is still considered the definitive work on JIT.

A Revolution in Manufacturing, the SMED System, by Shigeo Shingo, Cambridge, MA, Productivity, Inc.; Originally published as Shinguru Dandori in 1983, English translation 1985.
The title of this book is an indication of the quality of the translation, but that was not a barrier to our plant. We studied the book cover-to-cover and got our introduction to Lean Manufacturing from this book.

The Goal, by Eliyahu M. Goldratt, First Edition Published in 1984, Second Revised Edition Published in Great Barrington, MA, 1992
This book is a business novel. It is the easiest book to read for an introduction to Lean Manufacturing and is definitely a classic. You will find this book on the reading list of most Operations Management courses. Goldratt went on to develop the 'Theory of Constraints' and write several related business novels.

Out of the Crisis, by W. Edwards Demming; 1986
This is the book in which Demming outlined his famous 14 points.

The Demming Management Method, by Mary Walton and W. Edwards Demming; 1988
Probably the best book there is on Demming and his management approach.

Toyota Production System – Beyond Large Scale Production, Taiichi Ohno, published in Japanese in 1978 and in English in 1988 by Productivity, Inc.
This is an explanation of JIT by the inventor. The English version arrived after JIT had become widespread in the US.

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
This landmark book about the Toyota Production System was the first to associate the term 'Lean' with manufacturing.

Lean Thinking, by Womack, James P., and Daniel T. Jones, Simon & Schuster; 1996
A follow-up on the story of Lean Production, this book extends the concept of Lean throughout the enterprise. It shows how the "lean principles" of value (as defined by the customer), value stream, flow, pull, and perfection can be applied to all areas of the enterprise. (Software development is implicitly included.)