LEAN SOFTWARE DEVELOPMENT

Dasari. Ravi Kumar

1.0 Introduction

They may be clearly identified, but are poorly acknowledged. The problems of the software development planet are responsible for most of the project failures that force managements worldwide to put more rigid processes in place to ensure compliance. More stringent processes at each stage are making the whole process a "Concrete-Life jacket".

By using Lean Production/Manufacturing principles not only quality concerns and other issues can be resolved, but also a continuous improvement cycle can be built in to the process. This can help in improving the quality of the software solutions /products each time a new software solution/product is built.

2.0 Root Causes of Software Failure

In most of the unsuccessful projects the following factors are identified as the root Causes:

2.1 Frequently and Rapidly Changing Customer Requirements:

The problem with the conventional software development approach lies in the assumption that customer requirements are static and can be defined by a predetermined system. Because requirements do change, and change frequently, throughout the life of most systems, they cannot be adequately fulfilled by a rigid design. "Do It Right" has also been misinterpreted as "don't allow changes". If the changes are not allowed customers are not satisfied. If the changes are allowed the company will have problems in delivering the software in compliance with the project base line.

2.2 Centralized Decision Making

At large the Software development firms still follow the traditional "control and command" style in decision-making concerning the projects. Every time a decision has to be made it has to come all the way from the top of the project organization structure. This method actually increases the lead-time and makes the whole process rigid and slow.

2.3 Rigid Project Scope Management

Holding the project scope to exactly what was envisioned at the beginning of a project offers little value to the user whose world is changing. In fact, it imparts anxiety and paralyzes decision-making, ensuring only that the final system will be partially outdated by the time it's delivered. Managing to a scope that's no longer valid wastes time and space, necessitating inefficient issue lists, extensive trade-off negotiations and multiple system fixes.

However, as long as limiting a project to its original scope is a key project management goal, local optimization will flourish—at the expense of the project's overall quality.

2.4 Traditional Software Practices (Linear Development)

Most of the quality issues in the software components are also because of the linearity in the development process that does not allow the iterations and quality checks to occur before the components move to the next stage in the development cycle. So the development progresses even there are some quality issues/'Bugs'.

3.0 Lean Manufacturing Principles

The principles of Lean Manufacturing, which are summarized below, can be used as a framework, or a guideline to address most of the issues of software development world:

Add nothing but Value

1. Eliminate waste
2. Minimize inventory
3. Do it right the first time

Focus on those who add value

4. Empower those who add value
5. Practice continuous improvement

Pull value from Customers

6. Meet customer requirements
7. Pull from demand
8. Maximize flow

Optimize the value tributary

9. Ban local optimization
10. Partner with suppliers

4.0 Application of Lean manufacturing principles to Software development process ("Lean Software Development")

Basically the lean manufacturing principles can also be applied to the software development process to resolve the issues and to improve the process and obtain better results.

4.1 Eliminate waste

The first step in lean thinking is to understand what "value" is and what activities and resources are absolutely necessary to create the value. 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. The seven types of manufacturing wastes are illustrated below (refer: Table 1.0)

The Seven Wastes of Manufacturing
Overproduction
Inventory
Extra processing steps
Motion
Defects
Waiting
Transportation

Table 1.0

The seven wastes of manufacturing are also applicable to software development. The seven wastes of software development, based on the above framework, are illustrated below (refer: Table 2.0.)

The Seven Wastes of Software Development
Overproduction (Extra features)
Inventory (Requirements)
Extra processing steps (Extra Steps)
Motion (Finding Information)
Defects (Bugs not caught by tests)
Waiting (Waiting for decisions, including customers)
Transportation (Handoffs)

Table 2.0

The project teams should try to avoid these wastes, which are very commonly produced during the development process.

4.2 Do It Right the First Time (Incorporate Feedback)

"Do It Right the First Time" does not mean to "Freeze the Specs." On the contrary, product (and software project) specifications change constantly. Lean discipline demands instantaneous adaptation to changing market conditions, which is best affected with a flexible architecture that readily accommodates changes, monitoring techniques that detect errors before they occur, and tests that are designed before development begins.

The "Do It Right the First Time" rule has been widely used to justify the decision to develop a detailed system design before code is written.
The problem with this approach lies in the assumption that customer requirements are static and can be defined by a predetermined system. Because requirements do change, and frequently throughout the life of most systems, they cannot be adequately fulfilled by a rigid design.

If we acknowledge the axiom that customers may not know what they want at the beginning of development and that their needs might change midstream, we must incorporate a method of obtaining customer feedback during development. Instead, most software development practices include a complex "change control process" that discourages developers from responding to user feedback. Far from ensuring a quality result, these change-resistant processes actually get in the way of "Doing It Right."

Lean development employs two key techniques that makes the change easy. Just as Lean Production builds tests into the manufacturing process to detect when the process is broken, similarly Lean development must “build tests” at various stages of the development process. As development proceeds and changes are made, the unit and regression tests are run. If the tests don't pass, programming may be stopped until the problem is found and corrected. A comprehensive testing capability is the best way to accommodate change throughout the development process.

The second technique that facilitates change is "refactoring" (Improving the design with out changing functionality), or improving the design of existing software in a controlled and rapid manner. With refactoring, initial designs can focus on the basic issue at hand rather than speculate about other features that may be needed in the future. Later in the process, refactoring techniques can incorporate these additional features, as they are required, making it easy to accommodate the future if and when it becomes the present.

4.3 Empower those who add value

A basic principle of Lean Production is to drive decisions down to the lowest possible level, delegating decision-making tools and authority to the people "on the floor." Often when software development environments under-perform, the instinctive reaction is to impose more rigid processes, specifying in greater detail how people should do their jobs. Lean Production principles suggest exactly the opposite approach. When there are problems in manufacturing, a team of outside experts is not sent in to document in more detail how the process should be run. Instead, people on the manufacturing floor are given tools to evaluate and improve their own areas. They work in collaborative teams with the charter to improve their own processes and the links to nearby processes for which they are suppliers or customers. Their supervisors are trained in methods of forming and encouraging work teams to solve their own problems.

Lean development similarly gives priority to people and collaborating teams over paperwork and processes. It focuses on methods of forming and encouraging teams to address and resolve their own problems, recognizing that the people doing the work must determine the details.

Software development involves the handoff of information at least once (from user to programmer) and often more than once (from user to designer to programmer). One school of thought holds that it's best to transfer all such information in writing, but in

fact, a great amount of tacit knowledge is lost by handing off information on paper. A second school of thought believes that it's far more effective to have small collaborating teams work across the boundaries of an information handoff, minimizing paperwork and maximizing communication.

4.4 Create a culture of Continuous improvement

In many software development projects today, 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 changes and more over these process-documentation programs indicate excellence only when the documented process excels in the context of its use. So these accreditations have their own advantages and disadvantages.

Iterative development can effectively employ the Plan-Do-Check-Act method. During the first iteration, the handoff from design to programming or programming to testing may be a bit rough. It's OK if the first iteration provides a learning experience for the project team, because the subsequent iterations will allow the team to 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. A simple Plan-do-check-act principle can be followed:

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

Product/Solution improvement is also enhanced in the iterative process, particularly if refactoring is used. In fact, refactoring provides a tremendous vehicle to apply the

principle of continuous improvement to software development. However, we need an improvement model that can span more than a single project. We must improve future project performance by learning from existing ones. Here again, Lean Production can point the way.

4.5 Meeting the Customer Requirements

Philip Crosby defines quality as "conformance to requirements." The 1994 Standish Group study "Charting the Seas of Information Technology—Chaos" stated that that the most common cause of failed projects is 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 signoff prior to proceeding with system design. However, this approach to defining user requirements is deeply flawed. As discussed above in the “Do it right the first time” principle the process should have the provision for the customer to make changes. The Software compliance and User (Customer) acceptance testing must be done with reference to the customer requirements.

4.6 Pull from Demand

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 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.

4.7 Maximizing flow

In lean software development, the idea is to maximize the flow of information and delivered value. 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. Similarly in software development the idea is to eliminate as many documents and handoffs as possible. In the lean software development emphasis is to pair a skilled development team with a skilled customer team and to give them the responsibility and authority to develop the system in small, rapid increments, driven by the customer priority and feed back.

4.8 Ban local Optimization (Rigid Project Scope Management)

Project managers have been trained to focus on managing scope, just as in manufacturing production managers concentrated on maximizing machine productivity. However, Lean software development is fundamentally driven by time and feedback. In the same way that localized productivity optimization weakens the overall manufacturing process, so focusing on managing scope impairs project development.

Holding the scope to exactly what was envisioned at the beginning of a project offers little value to the user whose world is changing. In fact, it imparts anxiety and paralyzes decision-making, ensuring only that the final system will be outdated by the time it's delivered. Managing to a scope that's no longer valid wastes time and space, necessitating inefficient issue lists, extensive trade-off negotiations and multiple system fixes. However, as long as limiting a project to its original scope is a key project management goal, local optimization will flourish—at the expense of the project's overall quality.

4.9 Best Procurement Practices (Partner with Suppliers)

The high quality and creativity of the supply chain partnerships far outweighed the putative benefits of competitive bidding and rapid supplier turnover. Partner companies helped each other improve product designs and product flows, linking systems to allow just-in-time movement of goods across several suppliers with little or no paperwork. The advantages of this collaborative supply-chain relationship are lasting and well documented.

5.0 Conclusion

The lean production is a good metaphor for software development, if it is applied in keeping with the underlying spirit of lean thinking.

The proposed lean software development process principles are:

- Eliminate waste;
- Satisfy stake holders;
- Empowerment;
- Deploying Comprehensive testing;
- Deliver as fast as possible;
- Refactoring;
- Learn by Experimentation;
- Measure Business impact;
- Optimize across organization;

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

When applied to software development in the right context and spirit, these concepts provide a much broader scope and framework for improving software development process.

The simple tenets of Lean Production and Lean thinking have brought dramatic improvements in a myriad of industries. If applied to software development process as “Lean Software development”, these practices can make highest quality, lowest cost, shortest lead-time software development possible.

6.0 References:

- David Berger, Bill Cloke, Lean products start with Lean design
- James P. Womack, Daniel T. Jones et. Al (1990), The Machine That Changed the World: The Story of Lean Production.
- Jones, D.T. & Womack, J.P. (1996). Lean Thinking: Banish Waste and Create Wealth in Your Corporation,
- Mary Poppendieck (2002), Lean Thinking: The Theory Behind Agile Software Development.
- Journal of Software Development, Lean Software Product Development
- Todd Phillips, Building the lean Machine.