



# Agility through LEAN Driven Software Development

**WHITE PAPER**

**coMakeIT**  
**2007**

## Contents

Introduction .....	3
Non-Iterative Development Lacks Speed.....	3
LEAN Solution To Software Development.....	4
Set-Based Development Approach .....	5
Test Driven Development.....	7
Make It Flow .....	7
Conclusion.....	9
References.....	9

## INTRODUCTION

**Today**, the competitive market expects *rapid software development and delivery with quality, repeatedly*. Even if a software business is world leader in its industry, it is no guarantee of a future. Business must be constantly improving faster than its competitors to grow or even maintain profitability.

Where to improve and how to achieve best results?

**LEAN Manufacturing** is a successfully proven process management practice. One of the prime focuses of LEAN philosophy is implementing and improving 'flow' in process. Implementing flow unearths problems that were hidden in the process that always existed. Fixing those problems resulted in 'Waste Elimination', and when non value added activities are eliminated; it was clearly apparent that speed of the process increased dramatically.

LEAN Philosophy in manufacturing claims the benefits of

- Faster Time to Market (Speed)
- Quality at Source (Quality)
- Cost Reduction (Low-Cost)

If LEAN can do wonders in manufacturing, logistics and services, why cannot the philosophy be practiced in software development?

**LEAN Software Development** can be termed as translation of principles and practices of LEAN Production into software development to achieve the benefits of speed, quality and low-cost.

This philosophy was coined as an approach in 2003 by *Tom and Mary Poppendieck* in their book 'LEAN Software Development: An Agile Toolkit'.

This white paper gives an insight into enhanced software development using LEAN practices.

## NON-ITERATIVE DEVELOPMENT LACKS SPEED

The 1990 book *The Machine that Changed the World*, by James Womack, Daniel Jones, and Daniel Roos, describes how automobiles were built in the 1890s:

Different contractors used slightly different gages to make their parts. When the parts arrived in the final assembly hall, their specifications could be best described as approximate. Workers took the first two parts and filed them to make them fit perfectly. Then they filed a third part to make it fit the first two. And so on, until the entire vehicle --- with its hundreds of parts --- was completed.

There is a striking similarity of this automobile building approach with sequential software development approach. But current Manufacturing practices have evolved with better approaches to cope up with speed. It is disheartening to see software development methodology that is falling short to match speed by sticking on to conventional wisdom of development

Time-honored sequential approach to software development cycle was,



With this sequential approach to software development, statistics indicate that the time to market characteristic grossly lacks speed.

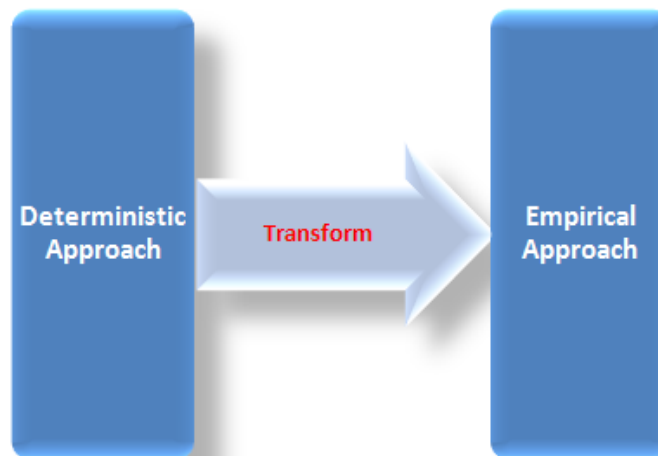
Speed, an important business component, coupled with quality and low-cost drives the sustainability of business in competitive market. *In current global economy, to be agile and rapid product delivery, repeatedly is a critical success factor.* This is unlikely to be accomplished with sequential approach.

Consistency in quality, rapid delivery with low cost is always a dream with conventional approach as it fails to identify and remove the non-value adding activities in its development value streams.

It is time that the conventional wisdom has to be crack opened and benchmarked industry proven best practices and philosophies be adopted in software development

### LEAN SOLUTION TO SOFTWARE DEVELOPMENT

Adopting LEAN solution approach is more about moving from deterministic school of thought to empirical school of thought



The much debated empirical approach starts with a high-level product concept and then establishes well-defined feedback loops that adjust activities so as to create an optimal interpretation of the concept. Empirical process allows software development adapt to changing environment

Anyone who has read through LEAN principles is very much aware about seven principles. Tom and Mary Poppendieck in their book have adopted these manufacturing principles to software development to maximize value and minimize waste.

7 Wastes – Manufacturing	7 Wastes- Software Development
Inventory	Partially Done Work
Processing	Paperwork
Over Production	Extra Features
Motion	Task Switching
Transportation	Handoffs
Waiting	Delay
Defects	Defects

Though this paper has listed up the mapping of LEAN principles to software, however the idea is not to discuss about these principles in detail. This white paper concentrates on the most proven LEAN approaches that enrich the value stream of iterative software development process.

- Set-Based Development
- Test Driven Development
- Make it 'Flow'

In a broader sense these LEAN approaches talks about achieving speed, quality at low cost.

#### SET-BASED DEVELOPMENT APPROACH

*“One of the most important things young recruits are taught is that when they were threatened, they should decide on the time box for a response, and not respond until the end of the time box.”-Military*

Customers would never stop changing their minds about their requirements. Changes can make the software development process to modify solution till deployment. There are two solution approaches to this.

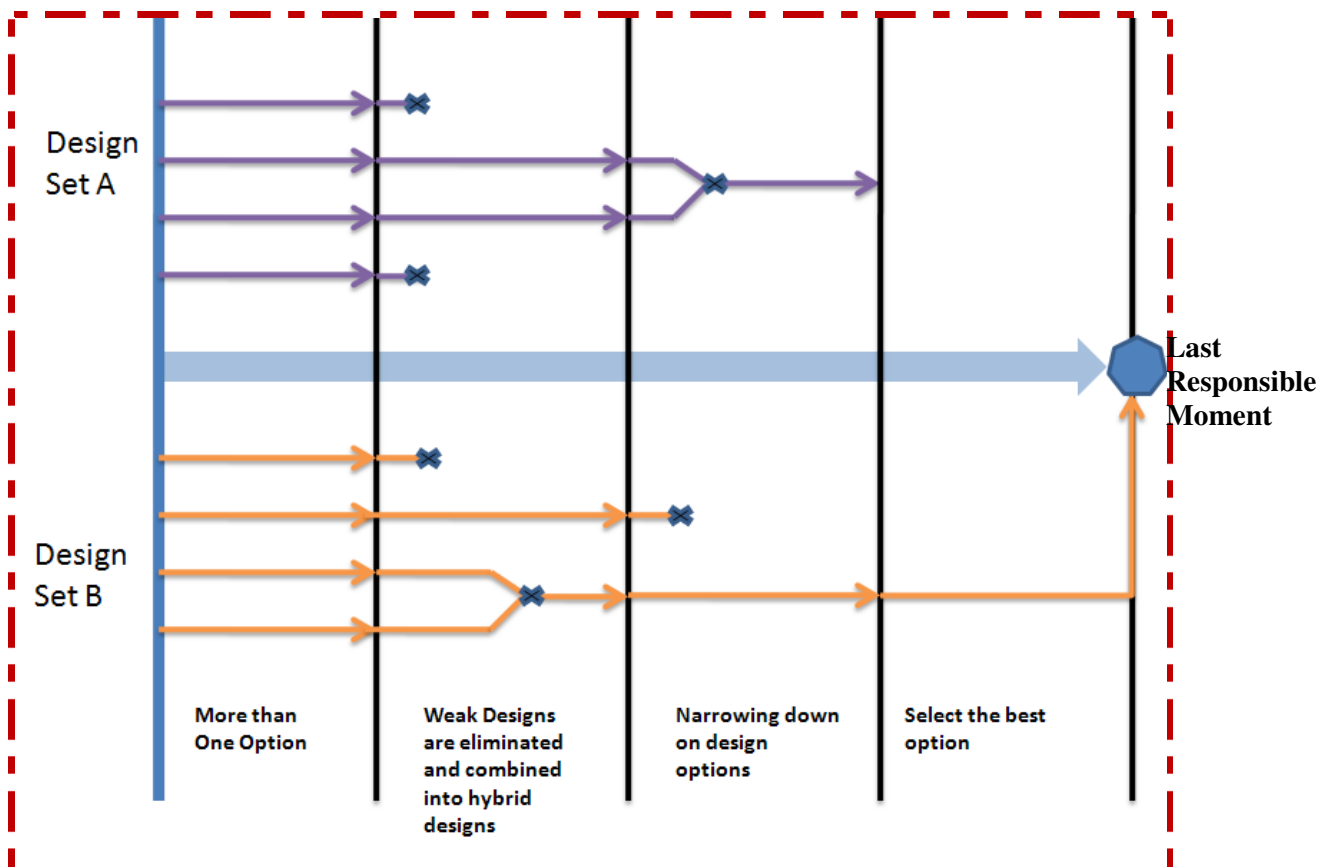
1. When customer orders something, it must be shipped so fast that the customer has very little time to change his mind.
2. Predict/anticipate changes and make your development process change tolerant

The first choice would work for *ebay.com*; it might not be very suited for software development. The second choice however works fine for software development, coupled with some LEAN thinking.

In Software development process, what is developed today may need a change because customer changed his requirements. This is where a need for anticipating changes comes in so that the development process, tolerates change.

Toyota followed an approach that made their process change tolerant. This approach is called 'Set-Based Design'. 'Set-Based Design' suggests that, for a problem, do not have just one solution approach but a set of solution approaches. For one of Toyota's first mass produced hybrid electric car 'Toyota Prius', the lead time was close to 15 months. To decide on what engine should the car be equipped with was a biggest challenge Toyota had. For initial four or so months Toyota maintained ten active engines under design and development. They never decided on one engine and tried to make it better. Rather they followed a radical approach to have a set of solutions. When they had to decide on the engine at the last responsible moment, they could actually make a quick decision because they knew which one among the ten was the best. So Toyota 'Deferred Commitment'.

Defer decision till the last responsible moment. Make more than one solution (Set) options. Until the commitment is made, gather information through feedback, analysis and learning. When the last responsible moment arrives, take the best of decisions among the options, with whatever knowledge is available till then. This is what set-based design is all about



Make all possible decisions as reversible, which means change tolerant. Those set of decisions that are irreversible, should be delayed till the last responsible moment. Since no decision is made, there is no question of changing the decision. However 'Defer Commitment' helps making business decision simple, because decision is among a set of solutions.

## TEST DRIVEN DEVELOPMENT

In software development there always exist a gap between design and implementation of that design. Software projects fail because of improper testing methodology. The conventional approach suggests 'Test at the End' approach. Business need 'Fail-Fast' approach to be productive with quality.

Test driven development is one of the LEAN approach which helps to bridge this gap. When a design is made, test cases should also be made parallel is the core idea.

### “Write executable specifications instead of requirements”

With TDD, before writing implementation code, the developer writes automated unit test cases for the new functionality they are about to implement. After writing test cases that generally will not even compile, the developers write implementation code to pass these test cases. The developer writes a few test cases, implements the code, writes a few test cases, implements the code, and so on.

The work is kept within the developer's intellectual control because he or she is continuously making small design and implementation decisions and increasing functionality at a relatively consistent rate. A new functionality is not considered properly implemented unless these new unit test cases and every other unit test cases ever written for the code base run properly. Implementation with quality needs testing to be aggressive and continuous. TDD believes in automation of testing, so that the involvement of human effort in the test cases has to be minimized.

TDD approach appears to yield code with superior external code quality, when compared with code developed with a more traditional waterfall-like model practice. The shortfall of TDD was the developers consumed more time to write the code as writing test cases parallel was also an activity. However end results have proved that TDD was an effective approach that improves programmers' productivity. Experimental results in comparison with traditional approach have proved that TDD approach facilitates simpler design and that lack of upfront design is not a hindrance. Transitioning to the TDD mindset is difficult. Delivering quality with close to null defects is a close to reality in this LEAN approach.

## MAKE IT FLOW

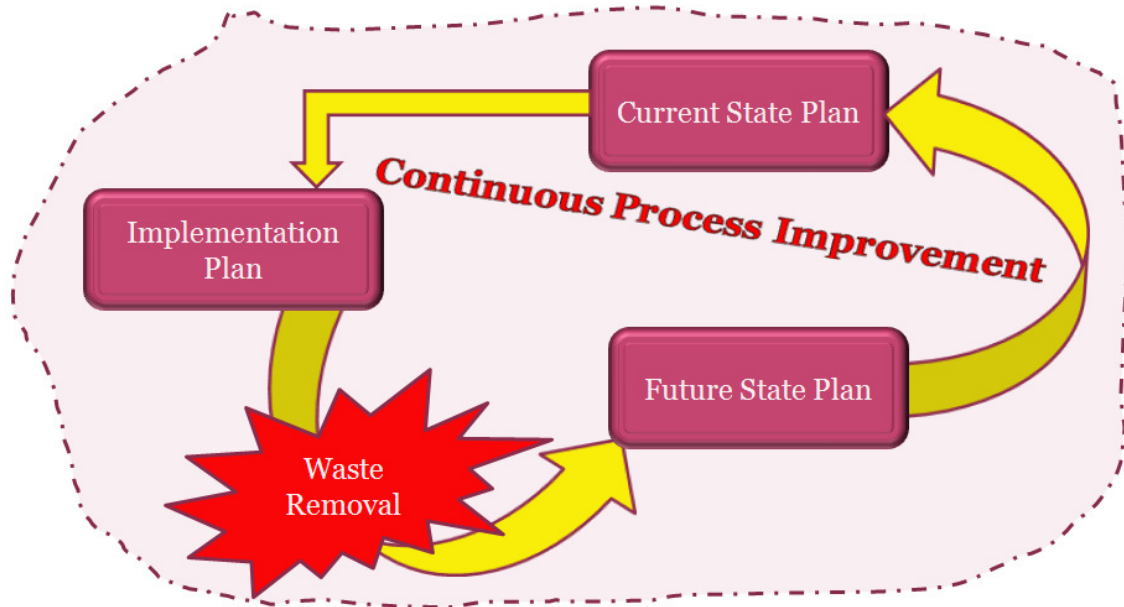
Value needs to flow from concept to delivery. If value needs to flow, all non value adding activities in the process stream have to be identified and eliminated. Agile focuses on people, value, and flow to deliver customer value and minimize waste. One of the tools that aid the value to flow is *value stream mapping tool*.

Value Stream Mapping is a 'Waste Visualization' diagnostic tool. It allows you to visualize the activities involved in delivering your product to your customer, from the perspectives of both the product and information flow.

**Current State Map** is a visual technique that pictures material flow, information flow with both the sequence and performance of value processes. It also highlights current *waste*.

**Future State Map** is the picture of an ideal value stream that is envisioned in which the wastes identified in the current state map have been eliminated.

**Implementation Plan** is the execution plan that is sequenced based on costs and benefits that will make the future state a reality.

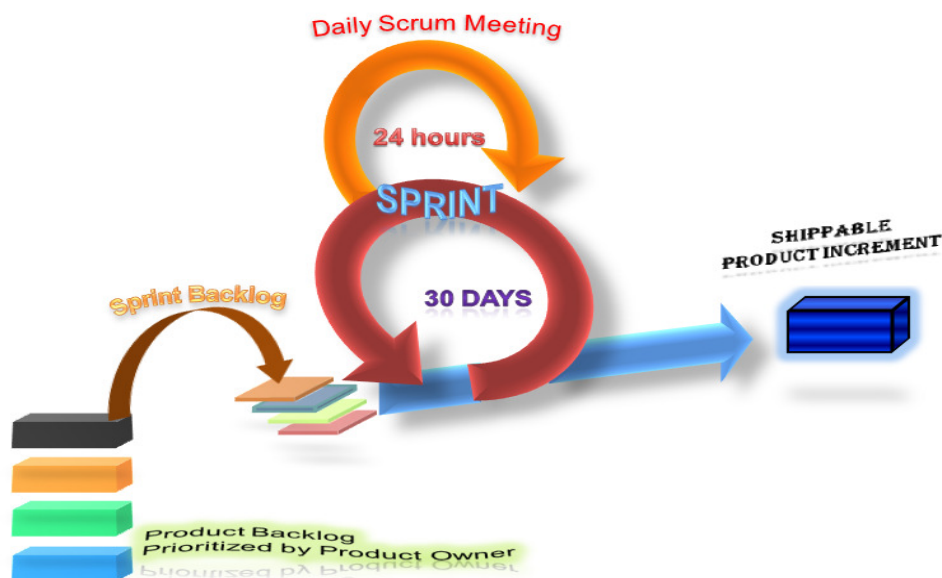


Value Stream Analysis helps implementing the first principle of LEAN, 'Eliminate Waste'. After making the value stream LEAN enough, the remaining six principles can be achieved by implementing the agile methodology of 'SCRUM'.

**“SCRUM is common sense that focuses on managing the system development process”**

A SCRUM process includes three phases: *pregame*, *game (development)* and *postgame*. It is in the game phase that SCRUM is most agile.

During the game phase of a SCRUM process, the system is developed in “sprints”. A “sprint” is an iterative cycle where the functionality is developed or enhanced. Each sprint will go through a traditional development lifecycle; however, these lifecycles are planned to last between one and four weeks.



SCRUM is a methodology that is feature oriented. The product features are broken down to tasks. These tasks are prioritized and this must be made transparent and communicated to development teams. SCRUM advises to work with features concurrently and integrate them once each of them is completed. There are no large-scale integration phases at the end of the project. Integration is a continuous process happening every few hours or on a daily basis.

If the new integrated code fails integration testing, the code on the integration machine is returned to the previous working version until the recent work has been fixed. Visibility is the crux of SCRUM. Problems are unearthed in the initial stages rather than giving surprises at the end. Changes made in development are reversible. For example, if a feature isn't completed within the time allotted, it can be taken out of the end product.

SCRUM emphasizes Refactoring. Refactoring is the process of making changes to the internal structure of the software so that it is easier to understand and less expensive to modify without changing its behavior.

Agile methods promote the concept of "collective ownership." Everyone owns all the code, and anyone can change anybody else's code.

SCRUM makes development iterative with proper feedback loop. It makes work 'FLOW'

## CONCLUSION

Speed with Quality at low-cost is achievable. The LEAN philosophy assures that. But the fruits can be reaped only if the LEAN principles in conjunction with some agile methodologies like SCRUM are practiced. This primarily needs an empowered team, right attitude and management support. LEAN principles help software development see a means to control costs, shorten development cycles and deliver applications that meet customers' needs.

Consequently, these organizations must be prepared into invest not only in an agile development method, but also in an advanced change and configuration management system that can provide the necessary versatility and control. Once done information and feedback are real time.

## References

Poppendieck. Book Page 19 Thursday, July 27, 2006 11:46 AM

The Agile Customer's Toolkit, Tom Poppendieck

LEAN Software Development, Mary Poppendieck

Competing on Basis of Speed, Google Video File

[www.LEAN.org](http://www.LEAN.org)

[www.agile.com](http://www.agile.com)

[www.agilealliance.org](http://www.agilealliance.org)