Lean Development – A team approach to Software Application Development

By

P. Nallasenapathi

Vice President, Saksoft

Date: March 2006

India	USA	Asia	Europe
Phone: +91 44 2461 4501	Phone: +1 212 286 1083	Phone: +65 6224 2550	Phone: +44 207 754 7154
Email: info@saksoft.com	Email: info@saksoft.com	Email: info@saksoft.com	Email: info@saksoft.com

TABLE OF CONTENTS

1. Introduction

Some of the best-known Software Development Methodologies are the Waterfall model, the Iterative Development model and the Agile Model (like Extreme programming).

During the boom of e-business applications Dr. Bob Charette came up with the **"Lean Development"** paradigm. According to the National Institute of Standards and Technology Manufacturing Extensions Partnership's Lean Network, Lean Development is "*A systematic approach to identifying and eliminating waste through continuous improvement, flowing the product at the pull of the customer in pursuit of perfection*[1]". But when applied to software development it translates to "*Lean Software Development reduces defects and cycle times while delivering a steady stream of incremental business value*[2]."

But putting the same in simple words, Dr. Bob Charette calls it **"Creation of change-tolerant software"**. We will deal with principles of lean development in this note, but it must be remembered that none of the principles are new in themselves, but their collation under one theme that is well focused, makes them particularly powerful.

2. Background

Lean Development (LD), has its origins in lean manufacturing, as was perfected by Taiichi Ohno (co-developer of the Toyota Production System). To get a better idea, we must understand that lean principles rely on 'reducing waste'. Let us examine some of the lean building blocks used in production systems and their relevance in software development.

Pull System – This is a technique for producing parts at customer demand. Manufacturers have historically operated using a Push System, building products to stock (per sales forecast), without firm customer orders. In the software context, practitioners tend to visualize the bigger picture and long-term view of the software application. This has traditionally delayed projects, as decision-making on technology components becomes complex, or end-users lose interest. Modular or phased implementations have advocated making applications on user demand, more incrementally and showing benefits all the way.

---

[1] Kilpatrick, Jerry. Lean Principles.

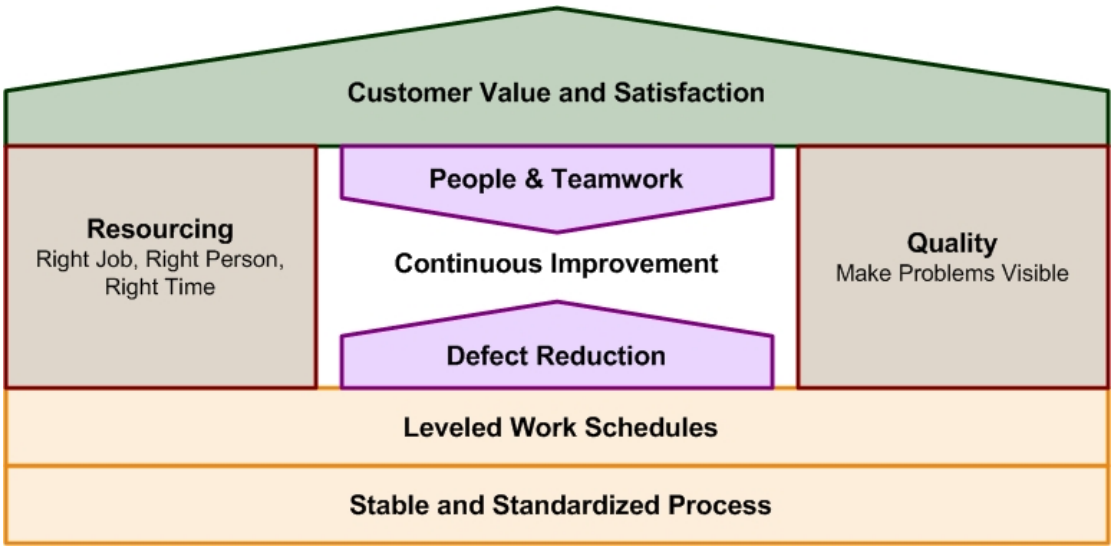
[2] Windholtz, Mark. Lean Software Development.

Work Cells – This is a technique of arranging operations and/or people in a cell (U-shaped, etc.) rather than in a traditional straight assembly line. Among other things, the cellular concept allows for better utilization of people and improves communication. Waterfall methodology tries to compartmentalize people into stages (those who gather requirements, the system analysts, the designers or architects, the coders, the testers etc.). This results in tending to rely on individuals (or skills of a coherent group) for excellence and quality. Teamwork, which works like a cell (complete in most if not in all respects of software development) has been used quite effectively in agile programming models.

Total Productive Maintenance – TPM capitalizes on proactive and progressive maintenance methodologies and calls upon the knowledge and cooperation of operators, equipment vendors, engineering, and support personnel to optimize machine performance. Agile programming or extreme programming tries to mimic such concepts by using feedback, rather than pre-planning, to ensure results are closer to expectations. The same can be extended to **Total Quality Management (TQM)** also. SEI CMMI compared to SEI CMM proposes a more integrated view to reduce defects and improve quality. SEI CMMI model stresses more on **overall capability** rather than individual exceptions.

Quick Changeover (a.k.a., Set Up Reduction and Single Minute Exchange of Dies) – This is a technique of reducing the amount of time to change a process from running one specific type of product to another. The whole concept of LD is to develop ”change-tolerant” software.

Batch Size Reduction – Historically, manufacturing companies have operated with large batch sizes in order to maximize machine utilization, assuming that changeover times were “fixed” and could not be reduced. Because LD calls for the production of parts linked to customer demand, the ideal batch size is ONE. However, a batch size of one is not always practical, so the goal is to practice continuous improvement to lean principles. In software many practitioners including proponents of LD stress on gradual development and deployment to help the user see early benefits.



3. Principles of Lean Development

Jim Hingsmith's perspective is easier for a non-technical team to understand. Some of Hingsmith's principles are explained here.

- **Satisfying the customer is the highest priority:** The development team must have practices to determine customer priority and to listen to the responses. The goal is maximizing ***customer satisfaction,*** not technical quality. Not meeting customer expectations is viewed as a failure.
- **Always provide the best value for the money:** Software should help solve customer problems or provide them a new opportunity at a reasonable cost. ***Value***, not perfection, is the goal. Value is a combination of product features that meet a customer needs at a specific time for a specific price. For example, in describing a typical German management culture, Womack and Jones bring out the fact that their strong technical expertise led many firms to "push ahead with refinements and complexities that were of little interest to anyone but the experts themselves"(1996). Sounds familiar.
- **Success depends on active customer participation:** Active participation is a ***collaborative joint effort***, not tokenism. Customer participation is more than just "buy - in"; active participation is essential to adapting to change and making real time, tradeoff decisions. As Charette commented, "No customer collaboration no LD."
- **Every lean development is a team effort:** ***Multidisciplinary teams***, rather than isolated individuals, are needed because diversity is so key to innovation and

fast cycle-time development. However the more diverse a team, the more difficult is to create an environment in which the team can jell.

- **Everything is changeable**: Mass development practices are based first on eliminating changes by defining requirements in the beginning and then by controlling any changes that happen to sneak in. Lean development assures that requirements changes will be continuous and that ***learning to adapt changes*** is a better strategy than trying to control them. The constant questions asked in a lean development effort are: What kind of charges could occur? What would we do if they occurred? How can we build the application to be more tolerant of this type of change?
- **Complete, don't construct:** "Buy rather than build" has long been a viable strategy for most application development groups. Components and templates are another dimension of using this principle.
- **An 80% solution today instead of a 100% solution tomorrow:** Markets are moving too fast to provide 100% solutions. It has been the experience of many software practitioners that the customers are almost always willing to give up functionality for early delivery of a working system. Furthermore, studies have shown that more than 45% of the functionality of software applications is never used.
- **Needs determine technology:** Choose the objectives of LD, then the technology to support it, not vice versa. The technology options today are so vast that it is easy to spend more time changing technologies than building business applications.
- **Product growth is feature growth, not size growth:** The critical factor in LD is ***delivering change-tolerant features***. When evaluating new features, the team should always consider how business practices might change and either affect or be affected by the software application. Size is not the issue.
- **Never push LD beyond its limits:** This last one might be labeled "understand the category of problem that LD is designed to handle." LD is not a silver bullet, it is a framework driven by typical business application software development problems. In a broad multi product (integrated) implementation scenario, a broad vision, detailed pre-planning, one-shot implementation are important and may contradict several principles enumerated above.

4. RADEC Model

RADEC stands for Saksoft's rapid application software development model framework. It inherits several conceptual foundations from Lean Development. SakC2C, Saksoft's program for application development is focused on complete solution delivery from **concept to completion.**

It begins with customer's problem statement. We begin from business problems and end with deploying and supporting the IT solution developed. Our software development process and OPTIMA (the SEI CMMI compliant Quality system) relies on many principles of LD, especially:

- The critical element in success is to get the application implemented
- Phased development and implementation to make users see early benefits
- Customer expectations are moving target – Saksoft seeks active customer participation for successful completion
- Saksoft believes collective team performance, delivery and end-result are more important than individual brilliance
- Technology is driven by Business needs

The idea behind RADEC model is to provide value that is derived by prudent combination of functionalities meeting customer expectations within a defined timeframe and cost.

5. The Lean Imperative

The gold standard of productivity is set by lean organizations: Toyota, Wal-Mart, Dell. Lean organizations concentrate on the rapid flow of value because they have discovered the fundamental principle of Lean: quality, speed, and low cost are tightly linked; in fact, they work together to provide world-class performance. Lean Development approach and Saksoft's RADEC understand and stands for that imperative. We can explain 4 broad imperatives in terms of benefits as,

High Productivity: 80% of the value of most systems is delivered by 20% of the features, and up to two-thirds of the features of most systems are rarely, if ever, used. If we would stop spending time specifying, designing, developing, testing, documenting and maintaining every little feature that is not going to prove particularly

useful to our customers, we could focus on providing them with a significant competitive advantage.

Rapid Response: The maturity of an organization is measured by the speed with which it can repeatedly and reliably execute its core processes. We must focus on delivering working applications step at a time, but functional enough to deliver value and advantage.

Superior Quality: The fundamental discipline of lean software development is testing. Tests define the requirements, drive the design, and document the system. Software should be built, integrated, and checked out with a test harness every day. Best-in-class organizations write the tests first, and deliver the test harness as a part of the code base.

Lasting Value: 80% of the code in any system is developed after first release to production. Some of the best software products have been evolving for a decade. Developers who have a deep understanding of the domain create robust, responsive software that stands the test of time.

6. Summary

Lean Development is an approach of **"Creating change-tolerant software"**. As explained earlier, none of the principles are new in themselves, but a collation of them under one theme that is well focused makes this theme particularly powerful.

In summary, LD focuses on customer expectations, productivity, and lasting value. LD enables teamwork and delivers applications that demonstrate benefits even as they evolve.