MECN 2028 – Lean Manufacturing

Lean Startup, Lean Software, SCRUM and Agile Methods

March 2015

Name	Student Number
Adrian Freemantle	1558184

Abstract

The principles of Lean Manufacturing have had a large impact on the way organisations approach and manage software product development. While there are large differences between manufacturing and the creation of software products, the core principles of Lean are just as applicable. The principles of eliminating waste, mapping the value stream, reducing cycle time, creating pull based systems, a continuous improvement culture, respect, just in time production and building quality are ultimately a mind-set. Frameworks like Scrum and The Lean Startup may help an organisation in becoming Lean, but only if these principles become part of the organisational culture.

Table of Contents

1 Introduction

In her book, *Lean Software* Development Mary Poppendieck states customers do not purchase software, instead they purchase a product such as a game, or a capability that enables a business process. Software development is therefore not a standalone process but forms a subset of product development. By looking at the principles implemented by Toyota Production System and Toyota Product Development System we can better understand how to achieve excellent software product development (Poppendieck & Poppendieck, 2003).

2 Lean Software Development

Although software development can be seen as a subset of product development, there are some significant differences between it and traditional product development. Software must be maintainable over long periods of time and be able to change as the customer's needs change. Value-adding and wasteful activities need to viewed differently when creating a software product as software development transforms ideas into working software that meets the customer's expectations (Poppendieck & Poppendieck, 2003).

Lean Software Development uses an empirical process which is intended to facilitate adaptation to change during the initial development process and while performing maintenance over the lifespan of the product. The empirical approach seeks to refine an initial product concept with *well-defined feedback loops that adjust activities so as to create an optimal interpretation of the concept* (Poppendieck & Poppendieck, 2003).

2.1 Seven Principles of Lean Software Development

Mary and Tom Poppendieck translated Lean Manufacturing principles into seven key principles of Lean Software Development in their book 2003 book *Lean Software Development: An Agile Toolkit:*

- Eliminate Waste
- Build Quality In
- Create Knowledge
- Defer Commitment
- Deliver Fast
- Empower the Team
- Optimise The Whole

2.1.1 Eliminate Waste

Muda, the Japanese word for waste, is used in Lean and TPS for activities that do not add value. While such activities are often easy to identify in a manufacturing environment, identifying waste in the activities of knowledge workers requires a different view on the systems and software development processes.

David Anderson, in his 2010 book *Kanban*, stated that software and systems development could be viewed as the process of converting information into working functionality where at the start of the process no information is available. The end product is created from the aggregated information received during the development process. In this view any

information *that moves us forward toward working functionality that meets the customer's needs is value adding-information"* (Anderson, 2010).

Therefore, any activities that are not creating value-adding information can be considered wasteful and should be eliminated or minimised. Anderson referred to such wasteful activities as costs and classified them as transaction costs, coordination costs and failure load (Anderson, 2010).

Transaction costs refer to activities that are exclusively performed for the purpose of preparing for the execution of value adding activities. Examples of such costs include requirements analysis, estimation, resource planning, audits, training and project planning. While these activates are unavoidable they do not directly add value. Minimising the amount of time spent on them would reduce the cost of the final product.

Coordination costs are incurred whenever two or more people are required to work together. Examples of such costs include sending emails, phone calls, status update meetings and delays in handing over work to different teams. As with transaction costs, coordination costs are unavoidable but can be minimised by keeping status-update meetings as short as possible, collocating team members and by empowering teams to self-organise.

Failure load is *demand generated by the customer that might have been avoided though higher quality delivered earlier* (Anderson, 2010). The impact of failure load is that it interrupts flow by increasing arrival variability. Failure load represents an opportunity cost that is not being spent on perusing more profitable ventures or on delivering value to existing customers. This type of cost is avoidable by building quality into the system.

2.1.2 Build Quality In

As previously discussed, a lack of quality creates failure load but it also affects flow and creates waste through the need for repeated testing, defect tracking and the rework of incorrectly build software. Incremental development practices allow regular feedback to be received from the customer to validate that the product being developed meets the customer's expectations while Agile practices like pair programming and test driven development constantly verify that the system is built correctly (van Vliet, 2012).

Automation of repetitive and time consuming testing and deployment steps not only builds quality into the system but eliminates waste. Continuous delivery is an automation process that aims to reduce the time between taken from conception of a requirement to the delivery of working functionality. This is achieved through the automation of the build, integration, test and deployment elements of the software development lifecycle. This allows for the fast detection of failures and provides a consistent and repeatable deployment process as it is consistently being tested. Should the release candidate fail at any of the stages, it is rejected and will not be able to progress to the next stage. This increases the visibility of failures and encourages rapid fixing of the problem as no further progression can happen until the issue is resolved (Humble & Farley, 2010).

2.1.3 Create Knowledge

Software development and Lean Manufacturing differ in that *development is like creating a recipe, while [manufacturing] is like making the dish* (Poppendieck & Poppendieck, 2003). While skills can be standardised within a software development team, each team member may be working on solving a unique problem in a new and innovative way and may be more experienced in certain parts of the problem domain.

Keeping team members at 100% utilisation leads to the creation of knowledge silos as there are no opportunities for the transfer tacit knowledge between team members. Skill development stalls and over time the skill set of the team as whole become outdated which reduces the company's competitiveness Creating a learning culture within the organisation can be achieved by creating collocated, cross-skilled, multi-disciplined development teams and by providing slack within the value stream. In such an environment, team members are able to effectively transfer tacit knowledge and transfer skills through practices such as pair programming and code reviews (Waters, 2010).

2.1.4 Defer Commitment

Software development projects frequently deal with problem domains in which there is a large amount of uncertainty regarding what should be built and how to build it. Requirements volatility also poses a problem as the need for business agility becomes increasing important in order to remain competitive. Early commitment through detailed designs, excessive estimation and irreversible decisions such as the purchasing of costly Commercial-Off-The-Shelf (COTS) software can lead to an increase in failure load as new information becomes available during the development process (Waters, 2010).

Deferring commitment to a specific deliverable, technology or design until the last possible moment helps mitigate this risk. Change is inevitable and therefore a certain amount of rework will need to be performed. The cost and impact of this rework can be reduced by ensuring that the architecture for the product is extensible and flexible while practices such as Continuous Delivery can quickly uncover any breaking changes (Humble & Farley, 2010) (Waters, 2010).

2.1.5 Deliver Fast

Traditional software development methodologies have cycle times that can take months or years to deliver working functionality due to attempting to deliver a perfect product. This long cycle time creates problems in that business requirements change over time and in the quest for perfection the product can become over engineered. In today's business environment speed to market is often perceived as being more valuable than perfection however this pressure to deliver fast can result in excessive failure load.

This can be mitigated by reducing the cycle time of converting a requirement into working functionality. This reduces the risk of changing requirements due a change in market conditions (Poppendieck & Poppendieck, 2003). Reducing the cycle time requires people with the knowledge of how to do the work, working as an effective team, constantly eliminating waste, building quality in and keeping the technical solution as simple as practically possible (Waters, 2010)

2.1.6 Empower the Team

Due to fast cycle times and deferred commitment, teams must be empowered to make technical decisions as a centralised command-and-control approach would lead to delays as team members wait for decisions to be approved.

> *When equipped with the necessary expertise and guided by a leader, they will make better technical decisions and better process decisions than anyone can make for them. (Poppendieck & Poppendieck, 2003)*

Self-organising teams can be viewed as Complex Adaptive Systems (CAS); Systems that are composed of agents (team members) that are constantly *forming new emergent structures with new emergent behaviours* (Myburgh, 2014). This requires a new form of leadership in which simple rules provide boundaries and constraints that allow the team to self-organise and adapt to their environment and find the appropriate balance between chaos and order. Such a leadership model encourages creativity and innovation by respecting the skills and abilities of the team members (Myburgh, 2014).

In his book *Kanban,* Anderson provides a sample set of rules and constraints that can be applied to help encourage self-organisation and the emergence of Lean behaviours within a team:

1. Visualize the Value Stream
2. Limit Work-in-Progress
3. Measure and Manage Flow
4. Make Policies explicit
5. Use models such as A3 to recognise improvement opportunities

2.1.7 Optimise The Whole

Traditionally the software industry tends to suboptimise individual parts of the software-development value-stream rather than taking a systems approach and looking for improvements across the entire system. This problem is made worse when Key Performance Indicators are applied to individuals or specific functions. This results in localised improvements, increased lead times, overproduction, increased failure load and higher transaction costs. A solution to this is to measure cycle time across the whole value stream (Poppendieck & Poppendieck, 2003) (Waters, 2010).

3 Agile Software Development

Agile Software Development, or Agile, refers to a variety of software development methodologies that originated in the late 1990's which all share a common philosophy around how software development should be approached; *close collaboration between the development team and business stakeholders; frequent delivery of business value, tight, self-organizing teams; and smart ways to craft, confirm, and deliver code* (Agile Alliance, 2015).

The important thing is not the tool you start with, but the way you constantly improve your use of that tool and expand your toolset over time – Mary Poppendieck, 2010

Agile has been influenced by the philosophy of Lean Manufacturing from its inception with both Mary and Tom Poppendieck being founding members of the Agile Alliance. Many of Lean Manufacturing's concepts such as Value Stream Maps, kanban systems, Work-In-Progress limits, reduced cycle time and reduced inventory are now core to most Agile approaches. Martin Fowler stated that Lean and Agile are so intertwined that they can't be seen as alternatives; *if you are doing agile you are doing lean and vice-versa* (Fowler, 2008)

3.1 Scrum

Scrum is an iterative Agile Software Development framework in which self-organising and cross-functional teams of five to nine people design, develop, test and release potentially deployable software every one to four weeks in close collaboration with the customer. This approach can be described as:

Instead of a large group spending a long time building a big thing, we have a small teams spending a short time building small things. But integrating regularly to see the whole. (Kniberg & Skarin, 2010).

Requirements are broken down into small concrete deliverables that can be completed by a team within a single iteration which usually lasts between one to four weeks. These deliverables are ordered by priority in a list called the backlog which is used to facilitate release planning. The backlog is not a static document and through regular collaboration with the client new deliverables are added while existing items are reprioritised as new insights are gained during the development process (Kniberg & Skarin, 2010).

At the start of an iteration a planning meeting is held where the client and the team select the next set of features to be implemented from the backlog. During this planning session the team and client will discuss acceptance criteria for the feature and resolve any uncertainty surrounding what needs to be delivered. The team may re-estimate the required effort based on new information which in turn may require a deliverable to be split further or deferred to a later iteration. A preliminary design is created and work is broken down into discreet tasks that need to be done to complete the development of the selected deliverables (Kniberg & Skarin, 2010).

During the iteration cycle, the team will meet once a day for a short status update meeting which is time boxed to a maximum limit of fifteen minutes. A team member called the Scrum Master coordinates and facilitates discussions while looking for ways to remove any obstacles that are preventing the team from making progress. Having a client representative attend in order to provide answers to any questions the team may have is ideal but not always practical (Kniberg & Skarin, 2010)

At the end each iteration cycle the team demonstrate fully working and tested code to the relevant stakeholders who decide if the agreed upon acceptance criteria have been met.

The software should be in a potentially releasable state as the client may choose to have the new functionality deployed; this requires that the quality of the delivered functionality must be production ready at the end of each iteration (Kniberg & Skarin, 2010)

The final part of an iteration cycle is known as a retrospective. This is a closed-doors meeting with only the team and Scrum Master attending. The retrospective is focused on enabling continuous improvement by empowering the team to critically evaluate their performance, identify opportunities to eliminate waste and identify opportunities to improve efficiency. All opinions are raised in an open and healthy atmosphere of trust and mutual respect with immediate feedback given on any suggestions made. The retrospective is considered to be the most important aspect of the Scrum development process as without it a kaizen culture will not develop (Kniberg & Skarin, 2010)

Through this process we can see how various aspects of Scrum align with the Seven Principles of Lean Software:

Lean Principle	**Effect of Scrum Principles**
Eliminate Waste	• Short iterations allow rapid feedback. • A feature set is "locked down" for a single iteration. • Daily meetings reduce the need for excessive coordination.
Build Quality in	• Deliver production quality software at the end of an increment.
Create Knowledge	• Daily meetings disseminate tacit knowledge. • Small teams ensure cross-skilling. • Small teams eliminate knowledge silos.
Defer Commitment	• What will be built is deferred to iteration planning. • Estimation for a feature is deferred to iteration planning. • Task breakdown and allocation is deferred to iteration planning.
Deliver Fast	• Potentially deployable software is delivered everyone to four weeks.
Empower the Team	• Retrospectives allow all team members to make suggestions. • Feedback on suggestions is immediately provided. • Daily meetings ensure that problems are raised early.
Optimise The Whole	• The full value stream is optimised though the inclusion of all stakeholders and the creation of cross-functional teams.

3.2 Kanban for Software

Kanban for Software is an Agile approach to change management. It is considered a more lightweight approach than Scrum as it does not prescribe any specific development or coordination practices but can be used in combination with Scrum or other Agile development frameworks. Kanban for Software's focuses on the Lean Manufacturing principle of limiting the amount of Work-In-Progress in order to shorten lead times, improve predictability and reduce waste. When used in combination with Scrum, iterations are often replaced with a pull based system that allows a constant flow of work to enter the system as capacity becomes available (Kniberg & Skarin, 2010).

With Kanban for Software, explicit Work-In-Progress (WIP) limits are set for each part of the software development value-stream. New work items, represented by a kanban signal

card, can only enter the system when existing work is completed or new work is pulled by a downstream part of the value stream. A work item or kanban card may only enter a portion of the value stream if its WIP limit has not been exceeded (Anderson, 2010).

By creating explicit WIP limits, demand is naturally balanced against capacity, bottlenecks are exposed and the impact of variance and waste on the value stream is made visible and explicit. By making these issues visible and encouraging a kaizen mind-set, teams are able to self-organise and implement improvements to their existing process. This introduces incremental change over a period of time without the resistance encountered by enforced change processed (Anderson, 2010).

4 Lean Startup

A startup can be defined as *a company working to solve a problem where the solution is not obvious and success is not guaranteed* (Robehmed, 2013) or a company that *at its heart is a catalyst that transforms ideas into products and services* (Ries, 2011). The high-risk nature of this business model is reflected by the fact that 90% of startups fail within 20 months of obtaining funding (Neil, 2015). CB Insights performed a post mortem on 156 failed startups and found that 42% of cases were due to a *lack of market need for the product* (CB Insights, 2016).

While Agile and Lean Software methods focus on ways of efficiently creating software for a customer, a startup is faced with the challenge of not having a specific customer or a defined specification of what their product or service should be. In 2011 Erich Ries wrote *The Lean Startup* in which he discusses his personal experiences in creating failed startups and how adopting a Lean approach to product development at IMVU, co-founded by Ries, helped him achieve success.

The Lean Startup approach reduces waste through a feedback loop that Ries refers to as Build-Measure-Learn. This approach allows for a hypothesis about what the customer wants to be tested as early as possible resulting in the company adapting according to feedback received. This is done by creating a minimum-viable-product which customers use. Through this use quantitative and qualitative feedback is obtained and used to determine which features to add or whether to change the direction of the product development. The faster a startup can cycle time through this loop the sooner they are able to adapt (Reis, 2011).

5 Conclusion

The principles of Lean Manufacturing have had a large impact on the way organisations approach and manage software product development. While there are large differences between manufacturing and the creation of software products, the core principles of Lean are just as applicable.

The principles of eliminating waste, mapping the value stream, reducing cycle time, creating pull based systems, a continuous improvement culture, respect, just in time production and building quality are ultimately a mind-set. Frameworks like Scrum may help an organisation in becoming Lean, but only if these principles become part of the organisational culture.

6 References

Agile Alliance, 2015. *What is Agile?.* [Online]
Available at: https://www.agilealliance.org/
[Accessed 03 March 2016].

Anderson, D., 2010. *Kanban.* 1st ed. Sequim: Blue Hole Press.

CB Insights, 2016. *The Top 20 Reasons Startups Fail,* s.l.: CB Insights.

Fowler, M., 2008. *Agile Versus Lean.* [Online]
Available at: http://martinfowler.com/bliki/AgileVersusLean.html
[Accessed 23 03 2015].

Humble, J. & Farley, D., 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* s.l.:Addison Wesley Professional.

Kniberg, H. & Skarin, M., 2010. *Kanban and Scrum - Making the Most of Both.* United States of America: C4Media.

Myburgh, B., 2014. *Situational Software Engineering: Complex Adaptive Responses of Software Development Teams.* s.l., Federated Conference of Computer Science and Information Systems.

Neil, N., 2015. *90% Of Startups Fail: Here's What You Need To Know About The 10%.* [Online]
Available at: http://www.forbes.com/sites/neilpatel/2015/01/16/90-of-startups-will-fail-heres-what-you-need-to-know-about-the-10/#25fbc51655e1
[Accessed 20 March 2016].

Poppendieck, M. & Poppendieck, T., 2003. *Lean Software Development: An Agile Toolkit.* 1st ed. s.l.:Addison-Wesley Professional.

Reis, E., 2011. *The Lean Startup.* s.l.:Portfolio Penguin.

Ries, E., 2011. *Creating the Lean Startup.* [Online]
Available at: http://www.inc.com/magazine/201110/eric-ries-usability-testing-product-development.html
[Accessed 20 March 2016].

Robehmed, N., 2013. *What is a Startup.* [Online]
Available at: http://www.forbes.com/sites/natalierobehmed/2013/12/16/what-is-a-startup/
[Accessed 20 March 2016].

van Vliet, H., 2012. *Software Engineering: Principles and Practices.* s.l.:John Wiley & Sons.

Waters, K., 2010. *7 Key Principles of Lean Software Development.* [Online]
Available at: http://www.allaboutagile.com/7-key-principles-of-lean-software-development-2/
[Accessed 20 March 2016].