Lean principles applied to software development – avoiding waste

Principiile Lean aplicate în dezvoltarea de software – evitarea pierderilor

NĂFTĂNĂILĂ Ionel
The Bucharest Academy of Economic Studies, Romania
e-mail: ionel@naftanaila.ro
BRUDARU Paul
The Bucharest Academy of Economic Studies, Romania
e-mail: paulbrudaru@yahoo.com

Abstract

Under the current economic conditions many organizations strive to continue the trend towards adopting better software development processes, in order to take advantage of the numerous benefits that these can offer. Those benefits include quicker return on investment, better software quality, and higher customer satisfaction. To date, however, there is little body of research that can guide organizations in adopting modern software development practices, especially when it comes to Lean thinking and principles. To address this situation, the current paper identifies and structures the main wastes (or muda in Lean terms) in software development as described by Lean principles, in an attempt to bring into researchers' and practitioners' attention Lean Software Development, a modern development methodology based on well-established practices such as Lean Manufacturing or Toyota Production System.

Keywords: *Lean, software development, agile methodologies*

Rezumat

În condiţiile economice actuale, o serie întreagă de organizaţii încearcă să continue trendul de adopţie a unor procese mai performante de dezvoltare a programelor informatice, pentru a beneficia de numeroasele beneficii oferite de aceste procese. Potenţialele beneficii includ o mai bună rată de recuperare a investiţiei, o calitate mai ridicată a programelor informatice, sau o mai bună satisfacţie a clienţilor. Totuşi, până la momentul actual există o serie limitată de cercetări care să ghideze organizaţiile în adoptarea practicilor moderne de dezvoltare de software, în special atunci când este vorba despre teoria şi principiile Lean. Pentru a remedia această stare de lucruri, lucrarea de faţă identifică şi structurează principalele pierderi (eng. waste sau muda în terminologia Lean) din dezvoltarea de software, într-o încercare de a aduce în atenţia cercetătorilor

şi practicienilor metodologia de dezvoltare de software Lean, bazată pe practici bine stabilite şi fundamentate, cum ar fi Lean Manufacturing sau Toyota Production System.

Cuvinte-cheie: *Lean, dezvoltare software, metodologii agile*

JEL Classification: M15

Introduction

Since he beginning of the current world financial crisis many technology-driven companies have suffered the effects, being forced to lay off people or drastically diminish costs (Wauter, 2009). The survival of the company itself becomes dependant of the time-to-market, deliver on time to the customer and minimize costs. The scientific literature abounds of examples in which the success of projects drive the success of companies, or, the other way around, the failure of a project puts the company out of business (Charette 2005), (Voas and Whittaker, 2002), (Jones, 1995). As a consequence, minimizing risk and approaching projects in a structured manner become critical success factors. Over the past few years software development organizations have learned about the benefits of Agile Methodologies, such as Scrum and XP. On the other hand, at the level of years 2008-2009 a researcher can identify a trend in the practitioners' literature (blogs, Internet sources, etc.) which shows an increase of the Lean methodology adoption efforts. As a consequence, while many organizations undertake significant efforts to implement Agile methodologies, the outlook of business consultants and project management practitioners in the field of software development extends to Lean practices.

However, little if none scientific research is to be found on the subject of implementing Lean software development methodologies in organizations. The most literature available is represented (with a few notable exceptions such as Poppendieck, 2007) by case-studies and anecdotal evidence, which, although a good starting point, needs to be extended by further, more systematic research.

Lean Development

Lean methodologies were not created with the special purpose of improving software development efforts. They usually address matters of increasing efficiency in production systems by eliminating waste and by implementing the "right" processes. As a general principle, the Lean methodologies consider the human resource as being the most flexible resource of the system; however, discipline is needed with regard to the moment when decisions are made. The Lean production systems consist basically of a process formed of five steps: defining value for the client, defining the value chain,

improving the value chain by "pulling", and continuous search for excellence (Womack and Jones, 2003).

In the IT literature there is a small confusion regarding the usage of term Lean. This term has been introduced in the Information Technology industry a long time ago, early versions of the Lean concept being built on top of Deming's team centric management concept, statistical quality control and process improvement (Deming, 1996), (Deming, 1993). These concepts have later been known as Total Quality Management (TQM) and the respective concepts have started to be used in a variety of activity fields. The Lean ideas have been incorporated as fundamental determinants of ISO or Six-Sigma standards, and as a consequence have been used in software development in a plethora of industries. Usually the software development projects have used a version of Lean based on quality, based on Deming's work on statistical quality control and continuous improvement (Scheer, 2005). These early initiatives have pushed the software development industry on a path of intensive measurements, statistical indicators, well-defined processes and large amount of documentation – a path which uselessly overloaded the software development budgets, without necessarily bringing productivity increase at the industry level.

Meanwhile, operations management (Bărbulescu and Bâgu, 2001), (Badea and Bâgu, 2006) has discovered the Toyota Model (Toyota Production System or TPS) which started with Deming's TQM but evolved independently between 1950's and 1970's. The Toyota Model became the reference model for what currently is known as Lean Manufacturing. Although its TQM roots are quite obvious, the base concepts of Lean manufacturing are quite different from TQM.

Lean product design represents a relatively new concept (Badea and Burdus, 2009). This methodology has been developed by the Toyota design studios, and represents the approach used with large success by Toyota for designing new car models. Toyota has adapted key concepts of Lean manufacturing to the environment of design studios, which is radically different from a car assembly facility. The Toyota success has lead to imitation and improvement trials from many industries, including the software development industry.

The Lean basic concept, which is avoiding overproduction, can be also found in the Lean development methodologies, but under unexpected forms. For instance, development of any artefact (function, procedure, class) which is not going to be consumed immediately can be considered overproduction. This also applies to requirements, use-cases, test plans, status reports and other artefacts which are regularly used in software development projects. For instance, when the requirements are over-detailed, the project's ability to adapt to change from client can be seriously compromised. The solution proposed by Lean methodology is to regulate the product of all artefacts by "pulling" them from the client. Lean methodology proposes to detail the requirements as late as possible (when the most things are known about the requirements) but in any case before they become necessary (Poppendieck and Poppendieck, 2006).

There are fundamental differences, though, between Lean manufacturing and Lean software development; for instance, Lean manufacturing is not placing a

significant importance over the moment when decisions are made; Lean software development, on the other hand, is very strict on this aspect.

A Lean initiative in a product development environment is centered on eliminating waste, and creating quality "from the first time". The techniques used in a Lean project, although sophisticated and quantitative, are not statistical by their nature. One cannot introduce statistical quality control over creative, development or designing processes. In this regard, Lean is very different from TQM.

The main seven manufacturing wastes, as identified by Shigeo Shingo (Shingo and Dillon, 1989) are:

- In-process Inventory
- Over-Production
- Extra Processing
- Transportation
- Motion
- Waiting
- Defects

By analogy, the main seven software development wastes, as identified by Mary and Tom Poppendiek ((Poppendieck and Poppendieck, 2003) are:

- Partially done work
- Extra Features
- Re-learning
- Handoffs
- Task Switching
- Delays
- Bugs

From all these wastes, *partially done work* is probably the most significant one. In Lean terms, this would be identified with work-in-progress, which essentially is waste, because until completed, the development team and the project manager will not know about quality issues, deployment on production issues, or customer satisfaction. Examples of partially done work can be: code that is completed but not checked-in on the version control systems, undocumented code, untested code (this refer to unit tests and functional tests), code that exists on the test environment but not on the production environment, code that is commented (Milunsky, 2009a).

Overproduction, as said above, is another significant waste that Lean addresses in the process of software development. In manufacturing, it refers to good or services that are not immediately needed or acquired by a customer. Basically it translates to inventory, which in turn translates into costs (as inventory can become obsolete, can be damaged, has storage costs, etc.). In software development, overproduction refers to features that are not really needed by users, or to "frameworks" which are supposedly going to make further developments easier, but will never be actually used. The reason for developing these never-to-

be-used features comes directly from the waterfall approach – which would force the product managers to think ahead all the necessary features for long-term projects – which in turn would lead them to anticipate users' needs and invest time, funds and energy into building software that is never going to be used.

The reason for which overproduction is wasteful is due to adding direct costs of development, but also indirect costs of maintaining a significantly more complex code base, introducing unnecessary bugs, creating poor-performing applications, etc.

Relearning is considered as being the third-most-important waste in software development. It refers to the time spent learning things that once were known by the development team, or the time spent to rework already completed features, due to poor code quality. Several examples of this type of waste:

- Undocumented code – if the developers won't document the code while it is fresh written, the code would need to be re-learned when subsequent natural changes are going to arise, or when bugs are going to show-up. Therefore, if the code is not properly documented, company could lose money and valuable time for re-learning.
- Poor planning – if project managers randomly assigns developers to features, each time a developer takes over a piece of code written by someone else, a natural learning process must occur; therefore, the company would lose time and money by allowing someone to learn details which are already known by someone else. There are situations when overlapping is to be considered best-practice, but this usually refers to critical sections of the application, and needs to be done in a well controlled manner.
- Poor quality – the most costly moment of fixing bugs is after the application has been deployed to production environment. This is mainly due to the fact that the developer has to re-learn the code (even if it's the same developer who initially wrote the code). Therefore, if the developer properly writes unit tests, and if the team takes the time to define proper acceptance test criteria, then the odds of reworking the code and consequently to relearn it diminish substantially.
- As demonstrated by Eliyahu Goldratt in Critical Chain (Goldratt, 1997), multitasking or task switching significantly increases development time, due to (along with other reasons) the developer having to re-learn the task at hand each time he or she switches back and forth.
- Poor communication and knowledge management is another factor which would lead to waste due to re-learning. However, in the modern days, with proliferation of wiki tools and other knowledge-sharing systems, along with search features, communicating between team members should not be a problem – at least from a technological standpoint. Proponents of Lean do not advocate a great deal of documentation, but instead a minimum set of notes over critical

development decisions, so that the initial developer or someone else taking over the job would spend as little time as possible in getting re-acquainted with the task.

Handoffs, in software creation, correspond to transportation processes in product manufacturing. Every time a developer delivers a piece of code to a different party, there is a certain loss involved in the process of knowledge transfer. Examples of hand-offs can be:

- A developer hands-off the code to a second developer. In this kind of situation, if the first developer did not document the code properly, the second one will have a very steep learning curve in trying to figure out the code already written. Moreover, he or she can make assumptions which might prove wrong, and therefore introducing unnecessary bugs in the system.
- A developer hands-off the code to testers. If Quality Assurance teams have no clue about what the software they are testing is supposed to do, and how it is supposed to work, they are likely to test for features which were never intended, or to overlook bugs in the very core of the application. It is important therefore that the developer properly documents the feature so that transition from one team to the other is as effective as possible.
- The development team hands-off the code to the client. An example of waste due to transition from the development team to the client is the increase in the number of support calls if the software is not properly documented and tested.

Practitioners of the field recommend a series of measures to counterbalance transportation wastes in software development (Milunsky, 2009b):

- Open communication between parties.
- Where needed, existence of proper documentation.
- Inclusion of all functional areas in the organization in the development process.

Task switching is a well-known and documented source of waste in projects in general, and in software development in particular. As shown above, each time a developer switches back and forth from one task to the other, a significant amount of time is wasted in order to re-learn the task at hand and to get into the flow of work. Matters get worse when a developer belongs to several development teams at once – situation which is fairly common; in this case, interruptions are more frequent, and therefore task switching occurs more often. E. Goldratt has shown in Critical Chain (Goldratt, 1997), that if, for instance, a developer starts concomitantly two projects, each of them with an estimated duration of one week, none of the projects will be finished in one week, whereas there is a significant probability that both projects will not finish in two weeks either. When the waste due to task switching is added, probably both projects will finish in about two and a half weeks. By comparison, if the developer would tackle only one project at a time, at least one of the projects will be done in one week, and

the other will be done after two weeks – and in addition there's no switching time to take into consideration.

It is usually difficult for managers to resist temptation to release more than one project in the organization pipeline. However, releasing too much work at once will slow things down, instead of increasing productivity (Goldratt, 1984).

One of the biggest wastes in software development in general is usually *waiting*, or *delays*. There are multiple types of delays in software development: waiting for someone else to finish their task, waiting for an approval, waiting for a project to start, waiting for a specialist to get hired or integrated in the project team, waiting for testers to provide feedback or waiting for the deployment team to do their part of the job.

The major problem with delays in software development is that they prevent the customer from obtaining the business value from the product as soon as possible. As a consequence, the speed at which the software development organization can respond to a new customer demand is directly proportional to the systemic delays within the organization's development process. Delays are therefore, from a Lean point of view, waste – and one of management's priorities should be to minimize these delays in the development cycle. One of Lean's most important principles, as shown above, is to delay decisions as much as possible, in order to make well-informed decisions; however, if decisions, once made, cannot be implemented rapidly, they can compromise the whole process of development.

Bugs – or *defects* – represent the most common-known source of waste in a software development organization. They represent waste not only taking into account the time spent by developers to find, isolate and fix them, but also the potential financial losses brought to the company as the result of malfunction. A critical bug identified early in the development cycle (ex. unit testing) is not a major waste. On the other hand, a minor problem identified only after the system is in production stage and users are already relying on the system can be a much more serious source of waste. From this perspective Lean software development completes very well with Agile methodologies such as Scrum and XP, which stress the importance of unit testing and continuous integration throughout the whole project development cycle (Beck, 1999).

Lean versus Agile

At its base, Lean represents a managerial approach to improving production systems. Lean is a methodology responsible for significant developments in productivity and quality over the last decades, and it is successfully used in industries which range from factories or logistics to pharmaceuticals or product development (Liker, 2003).

Agile, on the other hand, is extremely specific to software development projects. Agile facilitates productivity increase by raising the level of client responsibility, focusing on creating the software itself, and not on creating plans or documents. At its roots, the Agile philosophy is based on three things: it assumes

that the specifications cannot be established at the beginning of a project and uses iterations and client interaction in order to identify necessary functionalities. Secondly, imposes a very strict discipline from a quality control point of view; and thirdly, it depends on the existence of a professional team which can efficiently fulfill the key tasks.

Lean and Agile overlap with regard to the concept of taking in consideration the changes which occur late in the process. The older methodologies, such as cascade methodologies, are often criticized for their inability to adapt the changes which intervene on the lifecycle of the project; both Agile and Lean are specially designed to accommodate these changes. Lean is not only prone to adapt to this type of change, but also encourages taking major decision as late as possible.

Also, one important thing to consider with regard to Agile methodologies in general, and Scrum in particular, is that they are designed to focus the team on delivering only the most important features, in a just-in-time manner, which would help mitigate the risk of overproduction, described above.

A software project can be Agile without being Lean, or can be Lean without being Agile. There is no direct clear link between the two concepts; however common understanding leads to the fact that they complete each other very well in a software development organization (Poppendieck and Poppendieck, 2003).

Conclusions and further research

Lean Software Development is an emerging paradigm; while the researchers and theorists of software development processes have shown little interest so far for the principles and practices of Lean Thinking applied to this field, practitioners have already started to apply these principles.

The current paper analyses the applicability to software development of the seven main wastes proposed by Lean. The paper identifies and analyzes each waste type, by mapping the general types of waste to the particular processes of software development. While a series of blog posts and articles have emerged on the Internet in the latter period about the subject, there are basically no significant research papers, most of them being case studies and anecdotic evidence. Therefore, there is a strong need for more empirical studies in this field; from this perspective, the current paper can constitute the departure point, as it synthesizes and structures the most significant research contributions to-date. From a practitioner's perspective, the current paper can be used as a first step in implementing Lean Thinking in software development, by providing a comprehensive synthesis of the most significant sources of practical knowledge.

While one of the conclusions that can be drawn from the above analysis is that without doubt using Lean brings substantial benefits to the companies, the current paper also shows that the current state of research lack of studies which analyses use and implementation of Lean practices in software teams and organizations.

References

Badea, F., and Burdus, E. (2009). „Contributions on the Lean Management in the current evolution of a company". *Economia. Management,* Vol. 12 (1), pp. 168 - 179

Badea, F. and Bâgu, C. (2006). *Managementul Producţiei: Studii de caz şi proiect economic*. Bucureşti: Editura ASE.

Bărbulescu, C. and Bâgu, C. (2001). *Managementul Producţiei*. Bucureşti: Tribuna Economică.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*, US Edition ed. Extreme Programming Explained: Embrace Change.

Charette, R. N. (2005, Sep.). IEEE Spectrum. Retrieved from: http://www.spectrum.ieee.org/sep05/1685

Deming, E. W.(1996).*Out of the Crisis*. Cambridge, Massachusetts: MIT.

Deming, E. W.(1993). *The New Economics: For Industry, Government, Education*. Cambridge, Massachusetts: MIT.

Goldratt, E. M. (1984). *The Goal*. Great Barrington: North River Press.

Goldratt, E. (1997). *Critical Chain*. Great Barrington, MA: The North River Press Publishing Corporation.

Jones, C. (1995). "Patterns of Large Software Systems: Failure and Success," *Computer*, vol. 28, no. 3, pp. 86-87

Liker, J. (2003). *The Toyota Way*. McGraw-Hill.

Milunsky, J. (2009a). agilesoftwaredevelopment.com. Retrieved from: http://agilesoftwaredevelopment.com/blog/jackmilunsky/7-wastes-part-1-partially-done-work

Milunsky, J. (2009b). agilesoftwaredevelopment.com. Retrieved from: http://agilesoftwaredevelopment.com/blog/jackmilunsky/7-software-development-wastes-lean-series-part-4-transportation.htm

Poppendieck, M. and Poppendieck, T. (2003). *Lean Software Development: An Agile Toolkit*. Addison Wesley.

Poppendieck, M. and Poppendieck, T. (2006). *Implementing Lean Software Development: From Concept to Cash*, 1st ed. USA: Addison-Wesley.

Poppendieck, M. (2007). "Lean Software Development," in *Companion to the proceedings of the 29th International Conference on Software Engineering*, Washington, pp. 165-166.

Shingo, S. and Dillon, A. P. (1989). *A Study of the Toyota Production System: From an Industrial Engineering Viewpoint (Produce What Is Needed, When It's Needed)*, 1st ed. Productivity Press.

Scheer, T. (2005, Sep.). *Sphere of Influence Inc*. Retrieved from: http://sphereofinfluence.com/soiblogs/tscheer/archive/2005/09/19/159.aspx

Voas, J. M.and Whittaker, J.A., (2002). "50 years of software: key principles for quality," *IT Professional*, pp. 28-35.

Wauter, R. (2009, Jan.). *TechCrunch*. Retrieved from: http://www.techcrunch.com/2009/01/22/sad-day-for-microsoft-5000-laid-off-earnings-and-revenues-down/

Womack, J.P. and Jones, D.T. (2003). *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*. New York: Free Press.