(Löpnummer, hanteras av kurssekreterare) **2006:PM07**

The Toyota Software Way

Naeem Afzal Kayani

EXAMENSARBETE
Magister med inriktning mot programvaruteknik Institutionen för teknik, matematik och datavetenskap

EXAMENSARBETE

The Toyota Software Way

Sammanfattning

Ingen svensk sammanfattning finns då denna uppsats skrivits av en engelskspråkig student. Se "Abstract" for mer detaljer (på engelska).

Författare:	Naeem Afzal Kayani		
Examinator:	Dr. Samantha Jenkins		
Handledare:	Dr. Steven Kirk		
Program:	Software Engineering, 2006		
Ämne:	Software Engineering	**Examensnivå**:	Master
Datum:	June 2, 2006	**Rapportnummer**:	2006:PM07
Nyckelord:	Toyoyta way, Software principles, Lean software, Toyota Production System		
Utgivare:	Högskolan Väst, Institutionen för teknik, matematik och datavetenskap, 461 86 Trollhättan Tel: 0520-22 30 00 Fax: 0520-22 32 99 Web: www.hv.se		

The Toyota Software Way

Summary

Software projects have a bad history of failing. Software development is also a manufacturing business just like cars. Toyota being very successful car manufacturer is worth study for the sake of finding suitability of their principles with respect to software Engineering. Toyota has fourteen basic principles which form the foundation of their work

This paper is analysis and feasibility study of these Toyota principles for software industry.

Författare:	Naeem Afzal Kayani		
Examinator:	Dr. Samantha Jenkins		
Handledare:	Dr. Steven Kirk		
Program:	Software Engineering, 2006		
Ämne:	Software Engineering	**Examensnivå**:	Master
Datum:	June 2, 2006	**Rapportnummer**:	2006:PM07
Nyckelord:	Toyoyta way, Software principles, Lean software, Toyota Production System		
Utgivare:	Högskolan Väst, Institutionen för teknik, matematik och datavetenskap, 461 86 Trollhättan Tel: 0520-22 30 00 Fax: 0520-22 32 99 Web: www.hv.se		

Förord

Thanks to Almighty Allah for the blessings bestowed upon me and I dedicate this work to my loving mother and family as whole.

Författare:	Naeem Afzal Kayani		
Examinator:	Dr. Samantha Jenkins		
Handledare:	Dr. Steven Kirk		
Program:	Software Engineering, 2006		
Ämne:	Software Engineering	**Examensnivå**:	Master
Datum:	June 2, 2006	**Rapportnummer**:	2006:PM07
Nyckelord:	Toyoyta way, Software principles, Lean software, Toyota Production System		
Utgivare:	Högskolan Väst, Institutionen för teknik, matematik och datavetenskap, 461 86 Trollhättan Tel: 0520-22 30 00 Fax: 0520-22 32 99 Web: www.hv.se		

Innehållsförteckning

The Toyota Software Way

Naeem Afzal Kayani
Department of Technology, Mathematics and Computer Science
University West
P.O. Box 957, SE-461 29 Trollhättan, Sweden
e-mail: *naeem_afzal@yahoo.com*

Abstract

Software projects have had a history of failing in recent decades. Software is gaining importance in industry and it is about time that this problematic situation is resolved. The reason behind Information Technology (IT) having these problems, is lack of business sense in IT field. This paper takes Toyota, which is a famous manufacturer, as a case study and proposes if and how Toyota's philosophy is suitable for IT in its bid to ward off its chronic Software Engineering (SE) dilemmas. Toyota's production strategy was found to be beneficial for software industry.

1. Introduction

Cost and efficiency are important parameters in any organization's success. The Toyota Company is currently the world's most successful and wealthy motor vehicle manufacturing company, whose manufacturing and product development methods are now widely studied [1]. Toyota's way of work is based upon their philosophy that can be expressed in terms of fourteen principles. These fourteen principles are also the foundation of the Toyota Production System (TPS)[2] practiced at Toyota manufacturing plants around the world[1].

We grew up in the industrial Age. It is gone, supplanted by the Information Age [3]. Software industry is also manufacturing concern like Toyota. Software industry also faces similar production problems like maintaining high quality with low cost and finding hard to keep adaptability to change.

Different departments of Toyota work in close coordination with each other and it has been highly fruitful. The Toyota's approach can help the software industry develop its own modified way of operation suitable for SE. SE is inherently phase based, where output of one phase is input of the next just like different departments of Toyota so this work becomes more relevant and appropriate.

The thesis is an attempt to look into the Toyota's principles and their use in software field and how can it help software houses to adapt their software development operations. Thesis covers the principles of Toyota and their impact in software world generally but does not cover detailed roadmap of this revolutio n. Such detail is left out for firms to decide themselves depending upon their current system and its modification ability.

2. Background

The most visible product of Toyota's quest for excellence is its manufacturing philosophy, called the Toyota Production System (TPS) [4].

Toyota's principles are vital for their way of work and at the heart of this philosophy is the idea of Muda (waste) elimination. Waste is everything in production cycle that takes input and does not add value to the product. [5]

This waste tracking and elimination is exercised in all the work areas in Toyota. In any event, the trick is to find waste or muda. [6]

3. Methodology

Literature study method was chosen for this thesis but lack of related research hindered that. Jeffrey Liker's book "The Toyota way-

14 management principles" was studied as anchor reference to get idea about how Toyota performs. "Lean Software Development" was the primary source of information for the targeted transformation of software companies. The methodology followed was to search books and articles on Toyota and its way of work. And on the other hand, operations of software firms were studied with perspective of how the operations could benefit from Toyota's work. The traditional way of making a chain of articles from primary source to secondary source was followed in initial half of the research and it led to new sources like 'lean software development is it feasible?" by Sowmyan Raman led to "Toyota Production System" by Ohno. BUT this approach was not followed in later stages because of the specific nature of thesis and because it created more wayward prongs rather than sticking to the idea of Toyota-SE interaction. Main concentration during the work remained on the questions of 'What Toyota achieves from this?' and 'why or why not would these principles work for software industry?' Based on this criterion, other books and articles studied in quest for further knowledge. These secondary sources were selected on account of prospects of potential benefits of integrating the two areas. These secondary sources like "Principles of Lean Thinking" also pointed towards primary sources like "Lean software development" and "The Toyota way". In fact, many of the sources in the work form a close network of references by cross referring each other with the two aforementioned primary sources recurring.

Compendex, IEEE Xplore and Scirus electronic libraries were major search engines used for articles. Google particularly Google Scholar was used to find other web sources. Web resources were searched to get the topic overview and to see what different ideas are in the air. Later, books and articles were studied to build the case on solid foundation. Books helped in making the both Toyota and SE sides of the issue whereas articles helped in bridging these two together.

Argumentation for the case was coupled to compensate scarcity of relevant research material.

4. Implementation

4.1 Long term philosophy

Base your management decisions on a long-term philosophy, even at the expense of short-term financial goals [5].

The principle addresses long term thinking. This should be kept in mind and exercised even at the cost of short term goals.

Sacrificing the short-term benefits for the sake of long-term customer satisfaction is always beneficial. In particular, the trading of some short term risk for long-term risk—for example, trading cost sharing for effectiveness-based payments—may be a simpler and more direct way to induce effort in external software development [7].

In such a relationship, suppliers and customers no longer vie for their share of a fixed amount of value, but work together long-term to find opportunities to increase that value for mutual benefit [8]. Toyota also aims for generating value for customer. Generate value for the customer, society, and the economy [5] Software industry is typically a value-added industry so this focus of value creation should not be overlooked in software creating concern. Value added applications do not save money or time, but are aimed at achieving the users' business mission. [9]

4.2 Continuous flow

Create a continuous process flow to bring problems to the surface. [5]

Toyota's processes are designed such that no process is waiting for the other process's outcome to do its work. Once the processes are such connected then problems do not remain hidden and are easily identified. This prompts us to change our processes such that problems are easily and quickly identified and do not remain hidden for long. This quality management principle is applicable to software field alike. Quality management decreases production costs because sooner a defect is located and corrected, the less costly it will be in the long run. [10].

Once all modules of software project are interconnected, then the flow itself unveils the error. Grouping encourages coordination by putting different jobs under common supervision [11]. The rule creates a supplier-customer relationship between each person and the individual who is responsible for providing that person with each specific good or service [12].
With the steadily growing power and reliability of hardware, software has been identified as a major stumbling block in achieving desired levels of system dependability [13]. Therefore continuous flow creation to eliminate defects is beneficial for software production process and it takes us to little quality control cost i.e. Zero Quality Control [14]. Zero Quality Control makes it possible to eliminate defects entirely with simple techniques and at low cost [6]

4.3 Pull systems

Third Toyota principle is to avoid overproduction by delaying item replenishment till last possible time. Liker expresses it as "Provide your downline customers in the production process with what they want, when they want it, and in the amount they want[5]." Tracking customer demand shift to predict future inventory requirement is also an inference of this principle. Of course, Japanese build schedule is more accurate in the first place and can accommodate a customer specified order more easily because of the much quicker feedback from the customers [15]

There is a general trend of push methodology. And this often leads us to overproduction. Toyota suggests otherwise. Use "pull" systems to avoid overproduction. [5].
In a pull system the user, or consumer requests the content. The user pulls the content through the channel [16]. Pull systems use a mechanism called kanban, which was originally patterned after restocking grocery store shelves[17]. This approach is also called the supermarket approach. The "supermarket" approach allows the Toyota production System to respond to demand fluctuations with a minimum of stock. [6]. Now the flow of work becomes easier to manage. The interesting thing about pull scheduling is that it takes the manger out of the loop of having to tell workers what to do. The work is self-directing [18] This increases

the flow of software development and helps the second Toyota principle working.
The principle is synonymous to requirements engineering [19] in Software Engineering. Requirements Engineering are one of the most crucial steps in software development process [20]. Verification and Validation is phase of requirements engineering. Verification and Validation (V & V) is the name given to the checking and analysis process that ensure that software conforms to its specification and meets the needs of the customers who are paying for that software [21].

The figure 1 [1] shows that most of the problems software projects face fall in requirements area. That is because too often the developer does not truly understand or address the real requirements of the user and his environment. [20]

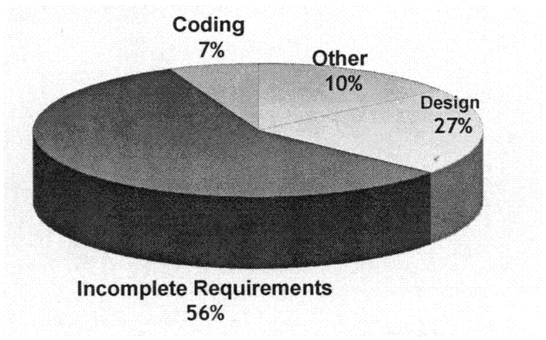

Figure 1: Sources of Error in Software Projects [1], Most of the software projects error are related with requirement elicitation

Therefore the focus should be on customer's demands and pull system rather than push ideology.

4.4 Workload level

Eliminating waste is just one-third of the equation for making lean successful. Eliminating overburden to people and equipment and eliminating unevenness in the production schedule are just as important [5]. During the last decades, various efforts have been made to improve working conditions [22]. Workforce's capacity is an organisational asset. The workload control of human resources is an approach to increase workforce flexibility [23]. The principal ingredient required to produce software is people [24]. Reduction of physical and mental workload is often pursued in industry [25]. Software Engineers need application specification to create software. "Ambiguity in specification" is a significant stressor in software engineers. This stressor is caused by the complicated communication process between users and engineers and by the shortage of tools or rules in this process. This stressor is considered to be the most typical workload which characterizes the software developing work [26].
Fourth Toyota principle calls for balancing the workload stress on software engineers. It is important to have workers identify the problems and develop ideas for improvement in collaboration with managers. [25]

4.5 Stop to fix problem

Build a culture of stopping to fix problems, to get quality right the first time[5]

This may look absurd to stop abruptly for every problem but it saves a lot of time and effort in all manufacturing areas. Information Systems implementers are not alone in facing the dichotomy between efficiency and effectiveness [9].

When defects are produced, and we find out only through data, we miss the chance to take appropriate corrective actions [27]. By missing the chance the quality that could be achieved is lost. Identifying and preventing systematic errors [28] can have a big impact on quality (in terms of defects) for a relatively small investment [29].

Quality is very important for every product. In service industry, a different perspective on quality has emerged [18]. Here quality has a new meaning. The service view of quality takes into account that every customer has a different idea of what constitutes a quality experience. In a service economy, quality does not mean conformance to a script; it means adapting to meet the challenging expectations of many different customers [30]. Quality is hampered by the problems we face during production.

Software problems are solved at many levels, by all members of the development team [18]. All the problems once faced should be properly documented to avoid further repetition. In order to solve problems that have not been solved before, it is necessary to generate information [18]. Thus, we can improve software quality by focusing on the prevention and early detection of defects, a readily measurable software attribute [29].

4.6 Stable repeatable methods

Use stable, repeatable methods everywhere to maintain the predictability, regular timing, and regular output of processes. It is the foundation for flow and pull [5].

The preferred method to make software project is iterative with repetitive customer feedback as it updates project team on regular basis. Successful projects involve customer feedback on a regular and frequent basis [31]. The principle is about repeatable processes or iterations to maintain predictability. Processes are flows by which raw materials become products [6]. Software industry can make such repeatable methods too. Software firms produce software. Software for customer is a group of features that he requires. A feature is something that delivers meaningful business value to the customer but is small enough that the team can confidently estimate the effort required to deliver it[18]. Software processes are very complex entities [21]. Sometimes it is not easy to make a feature in a repeatable iteration due to complexity of the task. If a feature can not be done in a single iteration, it should be broken down into smaller features [18]. It is hard to identify and harder still to deploy effectively. But once you find it, you win [3].

4.7 Visual control

Use simple visual indicators to help people determine immediately whether they are in a standard condition or deviating from it. [5]

Focussing on the VRAPS [32] principles can make hidden risks and opportunities of software architecture visible [33]. Apart from the architecture area, there is little number of visual indices known for software development. Software development does not involve much material things for visual indication. It is more a mental and algorithm based production industry than mechanical one. Therefore this Toyota principle does not hold for SE industry.

4.8 Reliable technology

Use only reliable, thoroughly tested technology that serves your people and processes [5]

We are witnessing an enormous expansion in the use of software in business, industry, administration, research, and even in everyday life [34]. This involvement gives rise to opportunism regarding the potential use of software in future industries and safe innovation is the key for such future investments. Many of today's great firms grew out of technological changes that they were able to exploit. Of all the things that can change the rules of competition, technological change is among the most prominent [35]. These added changes create many new features and dimensions in the software. As a consequence, software programs are becoming increasingly large and complex. [34]. Software now-a-days is a combination of many pieces of software. A piece of software is said to be reliable if it works and continues to work, without crashing and without doing something undesirable [36]. One loose link can break the whole chain. Therefore, despite all the lures embedded in the innovation philosophy, Toyota calls for safety first approach.

4.9 Grow leaders

Grow leaders who thoroughly understand the work, live the philosophy, and teach it to others [5].

One of the most significant factors governing economic relationships in the final decades of the 20th century was the recognition of the paramount role intellectual capital plays in creating and maintaining the competitive edge so necessary for corporate success [37]. Intellectual capital provides the necessary shock absorbing ability to any organisation.

In reality, unanticipated problems are a fact of everyday life for project managers. [21]. Successful project leaders apply a problem solving management style [38]. These leaders in the organisation must have digested the company philosophy towards commitment for excellence to forward it to fellow employees.

Furthermore, it is important to build the downward chain of forwarding the learning experience and company philosophy. All leaders, of course, will be responsible for training their constituents [39]. Software houses also face this situation and where new leaders can not fit into the organisational setup because they have different perspective than that of the organisation. A leader often has to confront issues and persuade others to do what they are unwilling to tackle [40]. Therefore software firms must build on leadership confidence in the company ideology and also be able to teach that to others.

4.10 People and teams

Develop exceptional people and teams who follow your company's philosophy [5]

The principal ingredient required to produce software is people [24]. It is the person that makes a software code. But this can not all be done on the individual basis. In today's working environments it takes a team to accomplish most purposes [18]. Team performance can be defined as the extent to which a team is able to meet established quality, cost, and time objectives [41]. It takes a team that stays together for a longer period of time and works with belief in what they are achieving together to improve their performance. … the team members must all contribute and support each other to be successful [Dyer 1984]

Software industry is becoming very vast now and people may work in remote places as virtual teams [42]. A successful cooperative relationship requires a strong cognitive belief that the partner is trustworthy [43]. Trust is a major factor influencing the cohesiveness among virtual team members [44]. Trust requires leadership to set and maintain values, boundaries and consistency [45]. Motivation and commitment is important and that can not be forced from outside. Improved performance comes from motivation, from arousing and maintaining the will to work effectively, not because the individual is coerced, but because he is committed [46]

4.11 Network of partners

Respect your extended network of partners and suppliers by challenging them and helping them improve [5]. Working with a network of partners requires keeping the whole network on their toes by giving them challenging targets. Firms need others to get all the needed factors of production [47].
Throughout the manufacturing sector, collaboration is gaining momentum [48]. Similarly, software firms need other companies for the work outsourcing [49]. Recently, IT outsourcing has been recognized as a strategy for increasing efficiency and cutting costs of the information systems implementations [51].
Information Systems (IS), a kind of Information Technology, are now developed with the involvement of part suppliers. When some companies decide that outsourcing is the preferred mechanism for achieving IS objectives, they often like to view their vendors as partners [51]. As the software projects are becoming huge, the factor of outsourcing becomes more important. Large systems are decomposed into subsystems that are developed independently [21]. Precise subsystem interface specifications are important because subsystem developers must write code that uses services of other subsystems [21].

But the ability of performing at the cutting edge is not common. The companies in the partner network should be efficient too to share the commitment for excellence. The larger problem of demonstrating efficiency must be addressed. [51]

This is also important that the software organization help members of the network in maintaining high level of excellence to reap benefits. Ultimately, the success of a collaborative engagement is directly linked to the relationship between the internal and external resources [48]

4.12 See problem yourself

Solve problems and improve processes by going to the source and personally observing and verifying data rather than theorizing on the basis of what other people or the computer screen tell you.[5]

Engineer should never be more than a stones throw away from the physical product [52]. The idea of going to the source to make correct decisions has specific importance for software engineering. Without visiting source of the problem, the corrected solution is likely to carry more errors. Open source projects generally have fewer defects than closed source projects since defects are found and fixed more rapidly [53].

No matter how much information is provided through data, it is difficult to see the true picture of workplace through data [27]. Data can never show all information necessary to make the right decision. The place where we can accurately capture the true state of the workplace is workplace itself. [27]

4.13 Decide slow, implement fast

Make decisions slowly by consensus, thoroughly considering all options; implement decisions rapidly [5]. Delaying design decisions in software development makes the system adaptive to subsequent changes.

The figure 2 [54] shows an example of the cumulative number of changes plotted by month for one of older systems [54]. A sharp rise means that software changes enormously throughout its lifecycle.

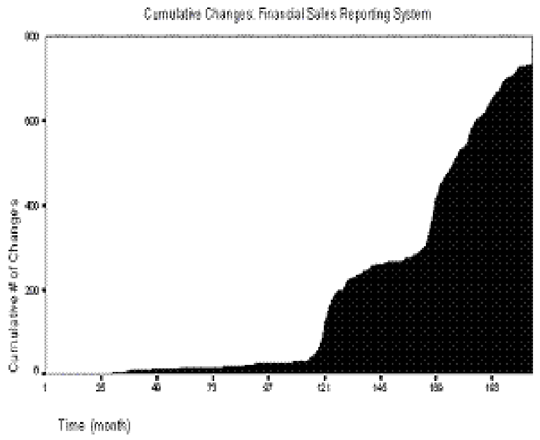

Figure 2: Changes in software with time [54], Cumulative number of changes(y-axis) increasing over time (x-axis)

The main reason software changes throughout its lifecycle, is that the business process in which it is used evolves over time [18]. Most business decisions are irrevocable; we usually don’t have the option to change our minds [18]. Therefore it is always better to delay decision making in such cases. Delaying irreversible decisions until uncertainty is reduced has economic value [18].

Nemawashi is the process of discussing problems and potential solutions with all of those affected, to collect their ideas and get agreement on a path forward [5]. All the options must be carefully scrutinized before decision making. Decisions are to be made based upon these options. Options allow fact based decisions on learning rather than speculation [18]. Changing software system architecture on late stages due to an earlier misunderstanding incurs heavy costs on the project. The system architecture affects the performance, robustness [55], distributability and maintainability of a system [21]

Delaying decision making obviously

gives us enough time to consider our decision in view of other available options and gives leverage to make necessary adjustments to the errors of judgement. Once decided, implementation can initiate at fast pace.

4.14 Keep improving

Become a learning organization through relentless reflection and continuous improvement [5]. Continuous business improvement requires all the people in the organisation to accept and welcome. This, in turn, requires the establishment of a new culture in which change and improvement are constantly occurring [56].

Japanese use term ‘Kaizen’ for continuous improvement. The message of the Kaizen Strategy is that not a day should go by without some kind of improvement being made somewhere in the company [56]. Software Process models are complex and keep evolving with time introducing new problems. The ideal process model must be modified dynamically as solutions to these problems are found [21]. Therefore the mechanics of performing the assessment should be restricted to the bare minimum needed to produce meaningful improvement plans [55].

5. Conclusions

Lean production provides better products in wider variety at lower cost [15]. But it is more a roadmap than a complete solution. Lean Software Development is not a development methodology but rather a way to think about whatever approach a team uses. [58]

The software operations are very similar in nature to the manufacturing. As shown in thesis all Toyota principles, with the exception of seventh pertaining to visual control, will thus be beneficial for the software development operations. Thus, the original goal of analyzing Toyota principles with focus on software firms was achieved. There be one possible improvement that could be done and it was not to take web for primary scope comprehension source as it distracted me off target and almost one month was lost in my quest for wrong questions.

6. Discussions

There were contemporary studies about idea based Toyota analysis i.e. with perspective of waste reduction etc. BUT thorough screening of all fourteen principles was missing. The thesis work has evaluated suitability of Toyota’s principles for software industry and consequently resolves the question of ‘whether Toyota’s principles are suitable for SE’. The thesis also provides information on which rules are suited to SE. Thus it provides foundation for software companies to make their own work principles.

7. Future work

This work would assist the management and decision makers in adapting structure and operations of their software companies to mimic Toyota’s way of work making them more adaptable to demand shift. Software companies can make their contingency plans and code for future demand shifts to gain adaptability like Toyota. Toyota follows its operations on the basis of its principles and software industry can move in this direction based upon work in this thesis. Software companies can build on it and make their own version of Software Toyota way i.e. their own set of principles according to their business.

8. Acknowledgements

I would thank Dr. Robert Feldt for his guidance throughout this masters program. I would also thank Dr. Steven Kirk and Dr. Samantha Jenkins for their encouragement throughout the thesis.

9. References

[1] Roy Morien, Agile Management and the Toyota Way for Software Project

Management , IEEE Intemational Conference on Industrial Informatics, 2005

0-7803-9094-6/05 IEEE

[2] [Online document], Available HTTP: http://en.wikipedia.org/wiki/Toyota_Production_System

[3] T. A. Stewart, Intellectual Capital: The New Wealth of Organizations , London : Nicholas Brealey, 1997

[4] The Toyota Way - the complete summary, [Online document], Available HTTP:

http://www.summary.com

[5] Jeffrey Liker, The Toyota way : 14 management principles from the world's greatest manufacturer, New York : McGraw-Hill, 2004

[6] Shigeo Shingo, The Shingo Production Management System, Cambridge, Mass.: Productivity Press, 1992

[7] [Online document], Available HTTP: http://portal.acm.org/ft_gateway.cfm?id=966391&type=html&coll=ACM&dl=ACM&CFID=15151515

[8] [Online document], Available HTTP: http://proquest.umi.com/pqdlink?did=379499071&sid=1&Fmt=4&clientId=39502&RQT=309&VName=PQD

[9] Meyer, Boone, The Information Edge , New York : McGraw-Hill, 1987

[10] William E. Lewis, Software testing and continuous quality improvement, London : Auerbach, 2004

[11] Mintzberg, Quinn, Voyer, The Strategy Process , Englewood Cliffs, N.J. : Prentice Hall, 1995

[12] Steve Spear, H. Kent Bowen, Decoding DNA of the Toyota Production Systems, Harvard Business review, 1999

[13] Chin-Yu Huang1, Michael R. Lyu, Optimal Testing Resource Allocation, and Sensitivity Analysis in Software Development, Accession Number: 87100165848

[14] Shigeo Shingo, Zero Quality Control: Source Inspection and the poka-yoke system , Stamford : Productivity, 1986

[15] James P. Womack, Daniel T. Jones, Daniel Roos, The machine that changed the world , New York : HarperPerennial, 1991

[16] [Online document], Available HTTP: http://en.wikipedia.org/wiki/Push_vs._pull

[17] Taichii Ohno, Toyota Production System, Productivity Press, 1988.

[18] Mary Poppendieck, Tom Poppendieck, Lean software development :an agile toolkit, Boston : Addison-Wesley, 2003

[19] [Online document], Available HTTP: http://en.wikipedia.org/wiki/Requirements_engineering

[20] H. Saiedian, R. Dale, Requirements engineering: making the connection between the software developer and customer, Information and Software Technology , 2000, DOI: 10.1016/S0950-5849(99)00101-9

[21] Ian Sommerville, Software Engineering, Harlow : Addison-Wesley, 2001

[22] DOI: 10.1016/j.ergon.2004.09.009 Istvan Balogh, Kerstina Ohlsson, Gert-Åke Hansson, Tomas Engström and Staffan Skerfving , Increasing the degree of automation in a production system: Consequences for the physical workload , International Journal of Industrial Ergonomics, 2006

[23] DOI 10.1080/00207540010022322

Franchini, Caillaud, Nguyen, Lacoste, Workload control of human resources to improve production management, International Journal of Production Research, 2001

[24] Eric J. Braude, Software Engineering- An Object Oriented Perspective , New York: Wiley, 2001

[25] Vink, Peeters, Gründemann, Smulders, Kompier and J. Dul , A participatory ergonomics approach to reduce mental and physical workload , International Journal of Industrial Ergonomics, 1995, DOI 10.1016/0169-8141(94)00085-H

[26] Y Fujigaki , R Kosugo, *Sangyo Igaku*, Tokyo, Mar 1992

[27] Alan Robinson, Continuous Improvement Operations- A Systematic Approach to Waste Reduction, Cambridge, Mass. : Productivity , 1991

[28] [Online document], Available HTTP: http://en.wikipedia.org/wiki/Systematic_error

[29] Learning from our mistakes with defect causalanalysis, D.N. Card, IEEE Software, 1998, DOI 10.1109/52.646883

[30] Prahalad and Krishnan, The new meaning of quality in the Information age, Harvard Business Review Article, 1999

[31] Martin, Agile Software Development, New Jersey: Prentice Hall, 2003

[32] VRAPS is abbreviations of Vision, Rhythm, Anticipation, Partnering and Simplification

[33] Dikel, Kane, Wilson, Software Architecture , London : Prentice Hall, 2001

[34] Crnkovic, Larsson, Building reliable component-based Software Systems , Boston : Artech House, 2002

[35] Michael E. Porter, Competitive advantage , New York : Free Press, 1985

[36] Douglas Bell, Software Engineering – A Programming Approach , New York : Addison Wesley, 2000

[37] Gary, Fane, Reza Vaghefi, Deusen, Louis A Woods, Competitive advantage the Toyota way, Business Strategy Review, December 2003, DOI: 10.1111/j..2003.00286.x

[38] Pressman, Roger, Software Engineering – A practitioner's approach, New York : McGraw-Hill, 1997

[39] Bower Marvin, The will to lead, Boston, Mass. : Harvard Business School, 1997

[40] Maurik Van John, Discovering the leader in you, London : McGraw-Hill, 1996

[41] Martin Hoegl, Hans Georg Gemuenden, Teamwork Quality and the Success of Innovative Projects: A Theoretical Concept and Empirical Evidence , Organization Science, 2001

DOI: 10.1287/orsc.12.4.435.10635

[42] [Online document], Available HTTP: http://en.wikipedia.org/wiki/Virtual_Teams

[43]Casper Lassenius, Maarit Nissinen, Kristian Rautiainen and Reijo Sulonen, The interactive goal panel: a methodology for aligning R&D activities with corporate strategy, Engineering and Technology Management, 1998. 0-7803-5082-0/98 IEEE

[44] [Online document], Available HTTP:

http://proquest.umi.com/pqdlink?did=298707881&sid=1&Fmt=4&clientId=39502&RQT=309&VName=PQD

[45] Duarte, Snyder, Mastering Virtual Teams, San Francisco, Calif.: Jossey-Bass, 2001

[46] Humphrey, PSP - A Self improvement process for Software Engineers, Boston : Addison-Wesley, 2005

[47] [Online document], Available HTTP: http://en.wikipedia.org/wiki/Factors_of_production

[48] [Online document], Available HTTP: http://proquest.umi.com/pqdlink?did=420958081&sid=2&Fmt=4&clientId=39502&RQT=309&VName=PQD

[49] [Online document], Available HTTP: http://en.wikipedia.org/wiki/Outsourcing

[50] [Online document], Available HTTP: http://proquest.umi.com/pqdlink?did=905476031&sid=1&Fmt=4&clientId=39502&RQT=309&VName=PQD

[51] Lacity, Hirschheim, Information SystemsOutsourcing, Chichester: Wiley, 1995

[52] [Online document], Available HTTP: http://www.sae.org/topics/leanfeb02.htm

[53] Paulson, J.W.; Succi, G.; Eberlein, A.; An empirical study of open-source and closed-source software products, IEEE Transactions on Software Engineering, April 2004, DOI: 10.1109/TSE.2004.1274044

[54] Kemerer, Slaughter, An empirical approach to studying software evolution, IEEE Transactions on Software Engineering, Jul/Aug 1999, DOI : 10.1109/32.799945,

[55] [Online document], Available HTTP:

http://www.webopedia.com/TERM/r/robust.html

[56] Barry Povey, Continuous Business improvement , London : McGraw-Hill, 1996

[57] Masaaki Imai, Kaizen: the key to Japan's competitive success, New York : McGraw-Hill, 1986

[58] [Online document], Available HTTP:

http://www.factbites.com/topics/Lean-software-development

10. Appendix:

Fourteen principles of Toyota way are [5]:

1. Base your management decisions on a long-term philosophy, even at the expense of short-term financial goals.

2. Create a continuous process flow to bring problems to the surface.
3. Provide your downline customers in the production process with what they want, when they want it, and in the amount they want.
4. Eliminating waste is just one-third of the equation for making lean successful. Eliminating overburden to people and equipment and eliminating unevenness in the production schedule are just as important.

5. Build a culture of stopping to fix problems, to get quality right the first time

6. Use simple visual indicators to help people determine immediately whether they are in a standard condition or deviating from it.

7. Use stable, repeatable methods everywhere to maintain the predictability, regular timing, and regular output of processes. It is the foundation for flow and pull.

8. Use only reliable, thoroughly tested technology that serves your people and processes.

9. Grow leaders who thoroughly understand the work, live the philosophy, and teach it to others.

10. Develop exceptional people and teams who follow your company's philosophy.

11. Respect your extended network of partners and suppliers by challenging them and helping them improve.

12. Solve problems and improve processes by going to the source and personally observing and verifying data rather than theorizing on the basis of what other people or the computer screen tell you.

13. Make decisions slowly by consensus, thoroughly considering all options; implement decisions rapidly.

14. Become a learning organization through relentless reflection and continuous improvement.