Lean Software Management: BBC Worldwide Case Study

Peter Middleton and David Joyce

Abstract—**This case study examines how the lean ideas behind the Toyota production system can be applied to software project management. It is a detailed investigation of the performance of a nine-person software development team employed by BBC Worldwide based in London. The data collected in 2009 involved direct observations of the development team, the kanban boards, the daily stand-up meetings, semistructured interviews with a wide variety of staff, and statistical analysis. The evidence shows that over the 12-month period, lead time to deliver software improved by 37%, consistency of delivery rose by 47%, and defects reported by customers fell 24%. The significance of this work is showing that the use of lean methods including visual management, team-based problem solving, smaller batch sizes, and statistical process control can improve software development. It also summarizes key differences between agile and lean approaches to software development. The conclusion is that the performance of the software development team was improved by adopting a lean approach. The faster delivery with a focus on creating the highest value to the customer also reduced both technical and market risks. The drawbacks are that it may not fit well with existing corporate standards.**

***Index Terms*—Agile, capability maturity model integrated (CMMI), development, lead time, lean, process, software, statistical process control.**

I. Introduction

LEAN thinking is important because it can reduce error rates to one per million units. It has been shown to have the potential to at least double the productivity of both manufacturing and service organizations. It also significantly reduces the time taken to deliver new products while substantially reducing cost. The evidence from Toyota (Japan), Porsche (Germany), and Pratt & Whitney (U.S.) confirms this [1], [2].

Applying the ideas from the Toyota production system (TPS) [3] or lean thinking to the management of software projects, therefore, promises great improvements. This case study records the practical experience gained between April 2008 and October 2009 by the London-based BBC Worldwide when it applied lean thinking for managing software development.

BBC Worldwide is the main commercial arm and a wholly owned subsidiary of the British Broadcasting Corporation (BBC). Its mission is to create, acquire, develop, and exploit media content and brands around the world in order to maximize the value of the BBC's assets for the benefit of the U.K. license payer. In 2008/09, BBC Worldwide generated profits of £103 million (U.S.$ 147 million) (before exceptionals) on revenues of £1.004 billion. (U.S.$ 1.4 billion) [4].

The basis of lean is the continuous elimination of waste [5]. This requires a focus on the flow of work through the system to ensure that material is produced only when it is needed and in the exact quantities required. This enables near-zero inventory levels to be approached, which makes production more flexible and also allows sources of defects to be quickly identified.

II. Literature Review

Lean emerged in 1990 as a term to describe the automobile production processes developed, since 1950 by Toyota. Toyota is generally regarded as the most efficient and highest quality producer of motor vehicles in the world. The term lean refers to the fact that Toyota was observed to use less space, manpower, materials, and time to make their products than their Western competitors [6].

The success of Toyota has generated a substantial literature on every facet of their approach. Schonberger [7] focused on how ''just-in-time'' production forces problems to the surface so enabling a habit of improvement. Imai [8] identified the ''Kaizen'' philosophy of continuous improvement as being the primary key to success. Rother and Shook [9] focused on value stream mapping as a process for identifying waste and envisioning a future state, thus enabling an organization to ''see'' their production from a new perspective.

Lean practices have been developed over the last 60 years. There is no pure lean approach as demonstrated by the different descriptions of lean in the literature, which identifies a range of overlapping lean principles. For example, Liker [1] has 14 principles, Womack and Jones [2] have five principles, and Shingo [3] also has five but different principles. Ohno's [5] focus was to reduce the time from customer order to product deliver by eliminating waste. Arguably he preached many principles, even though they are not laid out as such. The complexity of analyzing lean is due to the specifics of each lean implementation being context-dependent.

When Toyota was setting up a new plant in America, Liker and Hoseus noted that Toyota ''…were not interested in teaching us to copy. They were trying to teach us to think and act in the Toyota Way'' [10, p. xxii]. Therefore, for Toyota, it was more a philosophy of management combined with their experience of what was successful that was important.

For an organization to adopt a new philosophy and ''see'' their task differently is a major change management exercise.

Manuscript received February 10, 2010; revised June 26, 2010, August 24, 2010, and September 2, 2010; accepted September 8, 2010. Review of this manuscript was arranged by Department Editor J. K. Liker.

P. Middleton is with the School of Electronics, Electrical Engineering, and Computer Science, Queen's University Belfast, Belfast, BT7 1NN, U.K. (e-mail: p.middleton@qub.ac.uk).

D. Joyce was with the BBC Worldwide, London, W12 8QT, U.K. He is now with the ThoughtWorks, Melbourne, Vic. 3000, Australia (e-mail: dpjoyce@googlemail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEM.2010.2081675

The lean deployment literature, therefore, includes stories illustrating the hurdles [11], [12] and detailed practical guides [13]–[15].

Lean development of new products is significantly different to conventional Western approaches [6], [16], [17]. Key elements include heavyweight project managers, integrating systems, cultivating organizational knowledge, and long-term staff deployment patterns. The techniques of deliberately delaying decisions [18] and set-based concurrent engineering [19] also provide substantial benefits.

Software is more malleable and cheaper to distribute than manufactured products. Software development is therefore, somewhat different to conventional product development and production. However, at a higher level, the principles are still the same in any specific application area [1]. It has been more common in software development to draw on concepts from the Toyota Production System [3] [5], such as kanban, with fruitful results. But some core concepts from lean product development like front-end loading using set-based methods are particularly important for software development [17].

If user requirements, likely patterns of use, and technology performance are unclear, then good initial design is difficult [20]. An alternative to ‘‘paralysis by analysis’’ is to have a fast process that produces software deliverables quickly to respond, as understanding of the business increases. Lean offers a way to optimize and discipline this process.

Lean as an approach to management can travel globally, and when combined with lower labor costs in developing countries can be a major threat to Western jobs [21]. Workers have also experienced lean as a mechanism for capital to exploit and stress the labor force; therefore, it has not been universally welcomed [22]. Lean has been successfully applied to aerospace [23] and services [24], [25]. By focusing on increasing the speed from ‘‘order to cash’’ and the continuous elimination of waste, there is no theoretical reason it cannot be applied to any area [5].

The first recorded experiments with lean software development were by Middleton [26]. Microsoft reported how the lean mistake proofing of a software process eradicated whole classes of errors [27]. The U.S. Department of Defense (DoD) concluded lean techniques were the only way forward [28]. The Cummins Engine Company ‘‘. . .provides some compelling evidence that the ideas of lean manufacturing are indeed applicable, in principle, to software development.’’ [29]. The DoD concluded that ‘‘. . .shifting to lean principles improves cycle time reduction and overall quality in the software development process.” [30].

Lean software development is an evolutionary, incremental approach as advocated by Gilb [31]. The mathematical basis of lowering batch size to reduce lead time has been well described [32]. Although it has different intellectual roots, it has much in common with Agile software development. The Agile Manifesto, which was produced in 2001 [33] contains no references to lean. Agile was mainly a reaction against the document heavy, plan driven software development approaches that were frequently not successful. However, lean ideas helped provide a context and specific tools for the development of Agile. For example “scrumban” [40] is derived from kanban. The scrum itself is similar to Toyota’s small work groups with their daily stand-up meetings.

The use of statistical process control in lean software development means that it has the quantitative rigor required by Capability Maturity Model Integrated (CMMI) level 4 [34]. CMMI is important because is it mandatory for the U.S. DoD software contracts and has also been adopted by some large companies. After 20 years in existence, the independent evidence that CMMI leads to improvements in product cost, quality, and timeliness is ‘‘slowly accumulating’’ [35].

Lean software is, therefore, attractive to both defense contractors obliged to use CMMI and corporations who use lean to manage their manufacturing operations. They both also usually have a large proportion of the value of their products created by the software embedded in them. A lean software process can, therefore, offer them three benefits.

1) The statistical process control in lean software could allow them to quantify their software development process, which may enable them to achieve certification to CMMI level 4. This would allow them to bid for the DoD contracts.
2) Using lean for both their manufacturing and software development would provide a common approach and language, therefore, simplifying management of their operations.
3) If lean does quickly enable an intrinsically lower risk and more productive approach to develop software, then it will increase profits.

Timberline Software in Oregon in 2002 with 450 staff was the first recorded full industrial implementation of lean software development. They reported considerable improvement, but their focus on setting a common tempo or ‘‘Takt" time for software development based on creating similar-sized work units was not easy to implement [36].

Early lean software ideas were developed by Poppendieck [37] and Middleton and Sutton [38]. These books explored how lean thinking could be transferred from manufacturing to the more intangible world and different culture of software engineers. Specific techniques on how the concept of kanban could be applied to software were also developed [39], [40]. Note that the use of these methods is partly a metaphor rather than a direct copying. For example, kanban in factories literally is a binary signal to replenish an inventory buffer, based on what the customer has taken away. In software it performs a similar function, but more broadly displays information on the status of the process and potential problems.

Moving upstream and applying lean thinking to influence project selection and definition also creates great benefits [41]. The proceedings of the first Lean & Kanban Software conference [42] and the work of Shalloway *et al.* [43] show adoption is spreading. However, there is a clear need for more rigorous case studies of implementations.

III. Research Methodology

The research hypothesis was that the application of lean ideas would improve the capability of a software development process. In operational terms this meant that implementing lean practices, which included low work in progress (WIP), and pulling work into the process only when there was capacity, would show evidence of reduced lead times, error rates, and variability, while demonstrating continuous process improvement. The null hypothesis was that the application of lean ideas would have no or a negative impact on the capability of a software process.

The research method was for an experienced researcher to observe and write up the operation of the BBC Worldwide Webmedia Department's software processes. The seven visits to London of 2–3 days each took place between June and October 2009. These were supplemented by numerous phone calls and e-mails.

The advice that with case study research ''. . .close adherence to the data keeps researchers "honest"." was followed [44]. The pure positivist approach is that truth can always be discerned from untruth, and that the truth can be discerned either by deduction or by empirical support and by no other means [45]. The interpretivist model is that that an in-depth understanding of a phenomenon may only be gained by studying it in context from the participants' perspectives [46]. For this study, the position that both approaches are useful was adopted [47].

Triangulation, gathering data from as many different sources as possible to assist accuracy was used [48]. Data were collected from:

1) the most mature software development team, called Digi-Hub;
2) semistructured interviews with developers, project managers, business analysts, and managers;
3) walking through the operation of the kanban boards that visually displayed the flow of work so enabling it to be controlled;
4) recording the precise operation of the lean system;
5) observing the daily "stand-up" meetings where work allocations were discussed and agreed;
6) review of statistical analysis of the outputs from the system;
7) a brief review of the work of four other lean software teams was carried out.

This is an exploratory case study [49]. The research is asking knowledge questions focused on recording and understanding how the system was operating.

Lean is an integrated system and impacts on all aspects of an organization, particularly in its selection and development of people [10], [50]. This case study focuses only on a software development team, as this was not an organization-wide lean implementation. This meant the team had to work within their existing framework to adopt lean practices, where they had discretion in how to manage their own work, and try to influence other parts of the organization where possible.

Given the constraint of working within an existing framework rather than a lean organization, the following of Liker's 14 lean principles [1] were focused on.

1) *Principle 2:* Levels of WIP, e.g., requirements, designs, and code, were deliberately kept as low as possible to create continuous flow and bring problems to the surface. Process improvement and waste elimination were routine.
2) *Principle 3:* Work was "pulled" into the software development system only when there was capacity to work on it, rather than "pushed" in regardless of capacity available.
3) *Principle 4:* Level out the workload by working with users upstream in the process to try and smooth future demands.
4) *Principle 5:* Build a culture of stopping to fix problems. For example, fixing poorly structured legacy code that was hindering productivity and product reliability.
5) *Principle 6:* Continuous improvement and employee empowerment by actively looking for "blockers" and insisting all work was handled through the agreed process.
6) *Principle 7:* Visual controls were used extensively. This accounts for the reliance on kanban boards as they were ideal for making the intangible software process visible.
7) *Principle 8:* Ensure technology serves your people and process. "Autonomation" [5] where technology can prevent problems was used, for example, to enable the frequent release of software.

The principles of adopting a long-term orientation and creating a learning organization were invaluable within the team to guide their behavior.

IV. Reliability of the Data Collected

With a case study, there is a danger of bias in the data collected, which would undermine or destroy the validity of the results reported. The following areas were reviewed to identify any possible distortions in the data.

A. *Time Line*

The implementation of lean was started in April 2008. Due to the necessity to stabilize the processes and adapt to the changes, data collection did not start until three months later in August 2008. The data used in this paper refer to the 12 months from October 2008 to October 2009.

B. *Size and Volume of Work Started*

The underlying work did not change, but the way it was managed altered significantly. The approach was to identify the most valuable feature a customer needed and aim to keep each software unit being built as small as possible. This focus on Minimum Marketable Features (MMFs) [39] aimed to deliver the maximum value as quickly as possible.

Both the size and the volume of work allowed into the development system were greatly reduced. In the first five months, November 2008–March 2009 inclusive, 84 features were started of which 52% were classified as small. In last five months of the study, July 2009–November 2009, 64 features (24% fewer) were started of which 75% were classified as small (see Fig. 1).

C. *Complexity of Work*

The list of all the work undertaken was reviewed. It was clear the work was very varied and came from different sources. There was no evidence to suggest the complexity of the work required by customers had changed. However, by altering the focus to deliver the highest value, but in as small as possible features, the reduced size would make the complexity easier to handle. This "divide and rule" strategy in effect reduced the complexity of each unit of software being delivered. Once the structure of the legacy software had been improved and the automated tools were in place, the integration of these frequent small deliverables into the large body of live code was not observed to cause difficulties.

D. Governance Arrangements

The structure was:

1) Business Board (strategy and budget);
2) Project Board (detail and authorize specific work);
3) Product Owner (reconcile business and customer wants);
4) users requesting work (sign off work completed);
5) end users (200–300 people).

End users are those internal to BBC Worldwide. The digital assets created were ultimately used by millions of people.

This governance structure had been unchanged since before April 2008, but over the period of the study, it was reported that stricter identification of the business benefits was required before projects were authorized. This may mean projects are better thought through as regards return on investment, but at the technical level, the work required by the customers was unchanged.

E. Composition of Team

The team personnel were the same, since October 2008 with the same project manager. The data reflect the work of all the team, not just selected high performers. Also, all work carried out, including low priority and legacy improvements tasks, were recorded. All the members of the team interviewed reported their skills had improved over the 12 months.

F. Engineering Practices

Work to improve engineering practices started in April 2008, which involved the following:

1) test-driven development (unit tests);
2) automated acceptance testing (main suite completed April 2009);
3) source control software;
4) bug-tracking software;
5) decoupling–improving legacy software (April–July 2008)
6) MMF concept introduced April 2009.

This resulted in higher test coverage and releases increasing from monthly to almost daily. The same engineering practices were in place, but their use was consolidated and improved during the study period.

V. Digital Hub (Digi-Hub) Team

In 2009, the team had an annual operating cost of £1.5 million (US$ 2.2 million) and a development budget of £675K (U.S.$ 965K). It was made up of nine staff: project manager, business analyst, software architect, tester, lead developer, three developers, and a support developer. It was working on a mix of developing new software and software maintenance. The technology used was C#, .NET, MS SQL Server, and legacy connected service framework (CSF) code. The Project Board agreed priorities to be released over the next three months.

Month starting	Small	Medium	Large	
01/11/2008	11	5	0	
01/12/2008	5	13	2	
01/01/2009	7	5	5	
01/02/2009	3	0	0	
01/03/2009	18	9	1	Total: 84 features of which 52% small
01/04/2009	13	2	5	
01/05/2009	7	4	0	
01/06/2009	5	3	6	
01/07/2009	15	1	0	
01/08/2009	13	4	1	
01/09/2009	8	4	0	
01/10/2009	7	2	0	
01/11/2009	5	3	1	Total: 64 features of which 75% small

Fig. 1. Size and volume of work started.

In April 2008, all the stages of the development life cycle (value stream map) were drawn onto the kanban boards. All work at each stage was then recorded on cards and placed on the boards. This exercise immediately showed that there was more WIP and more bottlenecks of work than previously realized.

Restrictions on the amount of WIP allowed at each stage were put in place. The team determined the WIP levels by starting with their constraints. They had fewer quality assurance/testing staff and business analysts than software developers. They were now seen to be bottlenecks; therefore, the WIP limits were derived from how much work they could handle. This evened out the flow of work through the process. The WIP limits were then revised as constraints moved and more experience was gained.

To break the work down into smaller units that could be delivered more quickly, the concept of MMF was adopted [39], [40]. An MMF is a chunk of functionality that delivers a subset of the customer's requirements that is capable of returning value to the customer when released as an independent entity. These are then broken down into stories (new features) and then further into tasks, which are just "to do" items.

A. Office Layout and Work Flow

Office layout is a key component. In the Toyota production lines there are *andon* lights displaying the status of production at any time; the same idea can be applied to software development. Information radiators and kanban boards were placed all around the work space to ensure that progress on a project was completely transparent and available for all to see (see Fig. 2). This enabled team members to be self-managing.

The strategic direction and prioritization of work was still set by the Business and Project Boards, but the software team now had a much clearer idea of their capacity and current WIP. In the Digi-Hub project, two kanban boards (A,B) and four information radiators (C,D,E,F) were used and positioned as shown in the Fig. 2. The layout of the boards evolved as the projects and staff understanding progressed.

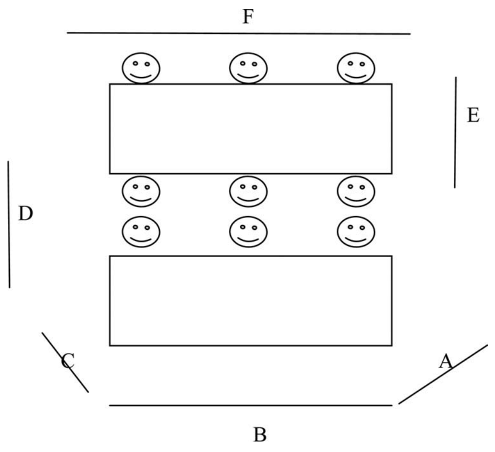

Fig. 2. Layout of kanban boards and information radiators.

It is important that work flows are kept as stable as possible. This is because sudden peaks and troughs of work are disruptive and will damage productivity. It is, therefore, necessary to try and influence the upstream work flows as much as possible. This information is captured on kanban board A: the ideation pipeline (see Fig. 3).

Any ideas or potential work from customers were recorded on a card and retained in proposed ideas' in case they trigger suggestions from the team. Any work, which is abandoned or has its priority changed is also recorded. Once the ideas have been clarified and broken down into small deliverable units, then they are ready to be "pulled" to the next board when the team has capacity to work on them. This kanban board B (see Fig. 4) tracks the progress of MMFs, Stories, and Tasks.

If there are problems in development (Dev.), these will quickly become apparent, as they will reach their WIP limit and become a visible bottleneck.

B. *Daily Standup*

The daily stand-ups last for about 15 minutes, and normally start at 10.15 A.M. each morning. They are carried out with all team members standing in front of kanban board B, which tracks the development phase (see Fig. 4). This is because this is where the bulk of the work is carried out. The daily stand-up is vital for the operation of the lean system. It is essential to facilitate the identification and removal of blockages and bottlenecks, and update the status and prioritization of work items.

The structure of the daily stand-up rhythm is given on information radiator C (see Fig. 5). First, everyone checks to ensure their work status is correctly displayed. Second, anyone who is "blocked," unable to progress due to something outside their control reports this, and appropriate action is decided to remove the obstruction. Third, any clusters of cards indicating a bottleneck are noted and the people reorganize to alleviate this. Finally, the work is reviewed to see if priorities have changed or if the work flow can be improved. There is an expedite work flow that can be used in exceptional circumstances to accommodate urgent items or if there is a high-priority late change.

There is no need or time in a large team for an individual report from each person. It is more effective to just flag problems to be resolved. The kanban boards make it clear to all the team the exact status of progress, blockages, bottlenecks and they also signal possible future issues to prepare for. This shared information enables the team to self-organize to ensure the work flows smoothly. The different colored cards used are listed below (see Fig. 5) and enable the exact status of all the team's work to be seen at any time.

No work was allowed to be carried out that was not recorded on the boards otherwise the system would be undermined. A typical "blocker" would be a slow or poor-quality delivery from a customer or supplier that the team member could not resolve. Another example would be a system that a developer had to interface with, but was having trouble obtaining passwords or technical data. Blockers could either be caused by a "special" one off or a repeated problem. The team would then confirm this was holding them back, explore alternative solutions, and if necessary, request a line manager with greater seniority and access to facilitate progress.

Information radiator D: Release notification and daily support process tasks board are used to ensure that any scheduled releases or other operations that have to be carried out during the week are visible. It acts as a reminder and check.

Information radiator E (Architecture, estimating, and breaking down projects): The board was used to record decisions on the architecture for the software, initial estimates, and how the work had been broken down into MMFs.

Information radiator F (Kaizen Board and Technical Debt): The team vote on which items they wish to work on to reduce technical debt or improve the lean system. Reducing technical debt involves work, such as improving poor legacy code or making a modification that could increase future productivity. Legacy software can be a severe constraint on current productivity. It is, therefore, necessary to explicitly reduce any technical debt by allowing time for improvements to be made, even though these are invisible to, and not requested by the customers.

VI. Performance Data on the Lean Software Systems

To evaluate the effectiveness of lean software management over 12 months of operation, some of the data used by the team are presented here. The team monitored the time taken for work to flow through various parts of the value stream. They also tracked quality and throughput measures using time-series statistical process control charts [51]. These charts have two important elements. First, the horizontal upper control limit lines show variance. The higher the line, the more variance, which severely damages productivity, is in the process. Second, the average is shown as the lower horizontal line over time. Much data used for management purposes are misleading unless presented in this way due to the statistical noise of natural variation [52].

A. *Lead Time*

Lead time is the total elapsed time from when a customer requests software to when the finished software is released to the customer. It is measured because it tracks how quickly and reliably software is delivered to customers. Lead Time is defined as the number of working days the work takes measured from

Proposed Ideas	Ready for decomposition	Decomposed Engineering Ready	Released
Waste			
On hold			

Fig. 3. Kanban board A: ideation pipeline.

Goal	Objective	MMF	Engineering Ready	Dev. Ready	Dev.	Dev. Complete	QA Ready	Engineering Complete	UAT	Release Ready
Work stream 1										
Work stream 2										
Work stream 3										
Work stream 4										
Expedite										

Fig. 4. Kanban board B: (development phase).

Key

Data	Card
Blocker	= Purple
Bug (in QA / UAT)	= Pink
Fixed delivery	= Small red (attached to card)
Expedite	= Small yellow (attached to card)
New Feature	= Yellow
Live Defect	= Red (bug found by customer, high priority)
Technical Story	= Green
Something missed	= White
Kaizen	= Blue

Stand up Rhythm (Daily)

1. Is the board correct?
2. Blockers
3. WIP / Bottlenecks
4. Work

Fig. 5. Information radiator C: kanban board and team performance indicators.

kanban board A: "decomposed engineering ready" (see Fig. 3) to kanban board B: "release ready" (see Fig. 4).

The main work requests are new features, which are a subset of an MMF. Other work would include technical features and live defects. "Decomposed engineering ready" means the customer has agreed to proceed and then the lead time clock starts. The items are then created for the engineering-ready input queue. Items are pulled into "engineering ready" only when capacity becomes available. The lead time clock stops when user acceptance testing (UAT) is complete and the items have reached release ready.

A reliable process will have low variance; therefore, a key objective is to continually reduce variance. Fig. 6 illustrates how the top lines, upper control limits, continually decline meaning that lead times are becoming more consistent and predictable. To show trends, the periods on the charts have been split from November 2008 to March 2009, April to June 2009, and July to October 2009. The results show software is being delivered with 47% less variance and on average 37% quicker.

B. Development Time

The Development Time measure gives insight into the efficiency of development. This portion of the value stream was directly under the team's control and not subjected to delays from upstream, downstream or third parties. It does not include engineering ready, quality assurance (QA), or related queuing times. Development time is recorded in working days, from kanban board B stages: Dev. ready to Dev. complete. The work units are either stories or tasks, which can be either standalone or part of an MMF.

The nine months for which data were available are shown on the statistical process control chart (see Fig. 7). The data has been split from February to March 2009, April to June 2009, and July to October 2009 to show trends. The declining upper control limits show variation in delivery times has reduced by 78% from 30.5 to 6.8. The mean time to develop fewer and smaller software features has declined by 73% from 9.2 to 2.5 working days.

C. Release Frequency Per Month

Release Frequency is defined as the number of items released to customers. An upward trend would be expected, as projects were broken into the smaller units and cycle time was reduced. The chart in Fig. 8 shows the number of releases per month increased by a factor of 8 from 2 in November 2007 to 16 in October 2009. There was a blanket release freeze on all

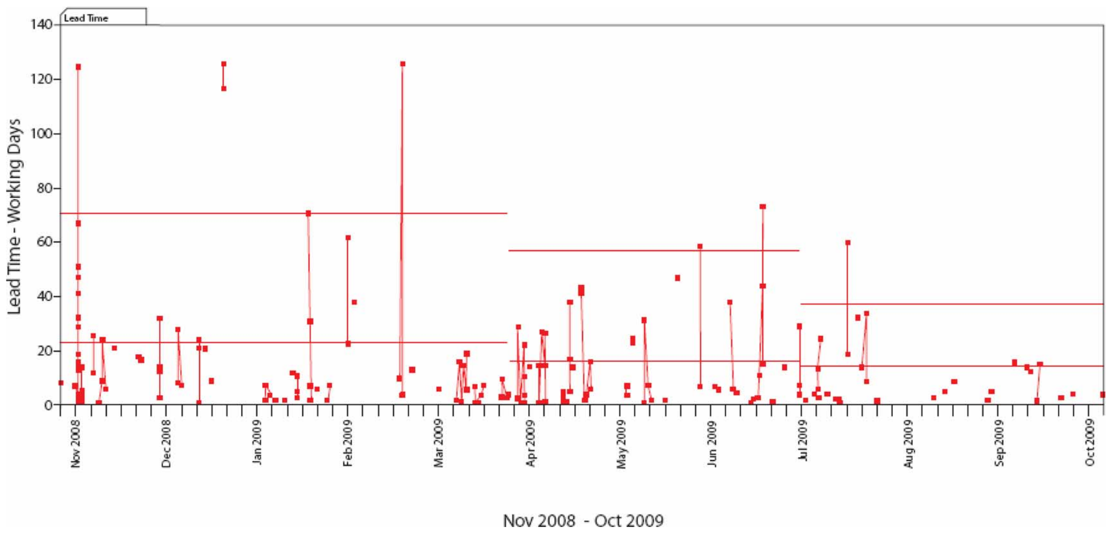

Fig. 6. Lead time: variance and average.

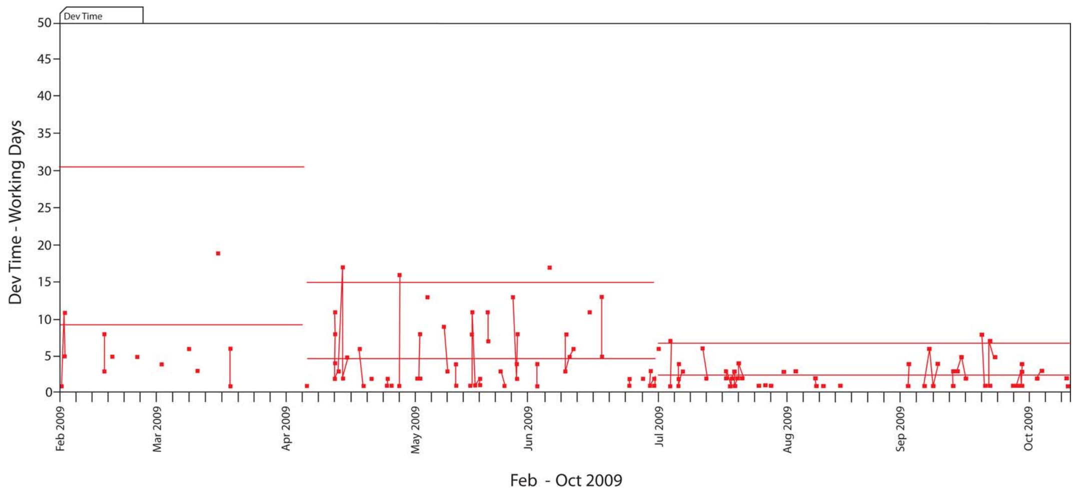

Fig. 7. Development time: variance and average.

releases in February 2009 to ensure complete priority was given to the production of year-end financial data, hence the drop in releases that month. The key issue of whether increased value was delivered is discussed later in the Analysis section.

D. *Live Defects Per Week*

Live Defects are the bugs reported by customers during a week plus the bugs still open. It is vital that the reductions in lead and development times are not at the expense of quality. Live defects are recorded on red kanban cards and added to the Dev. (Development) stage of kanban board B. The chart in Fig. 9 is split between October 2008 and June 2009 and July and September 2009. The reduction in variation indicates bugs were being fixed more quickly. The mean numbers of bugs open each week also slightly declined.

E. *Continuous Improvement Per Month*

The daily stand-up is concerned to identify and remove anything that is preventing progress. To do this, "blockers" are actively identified, assigned, tracked, escalated, and removed. This is a mechanism for making continuous improvement routine. Evidence of the effectiveness of this is shown in a statistical process control chart (see Fig. 10). The periods on the chart have been split from September 2008 to March 2009; April to June 2009, and from July to October 2009. The variance reduced indicating that problems with the process were being resolved more quickly. While the total number of problems identified increased, the average number of days work was "blocked" fell sharply.

VII. Analysis

The underlying type and complexity of work did not change significantly over the 12 months studied. The team and the governance structure remained comparable. The engineering practices were improved, but most were in place by October 2008, which was the start of the period studied. This was a stable team with better tools whose skill levels increased during the 12-month period studied. However, lean with its low WIP

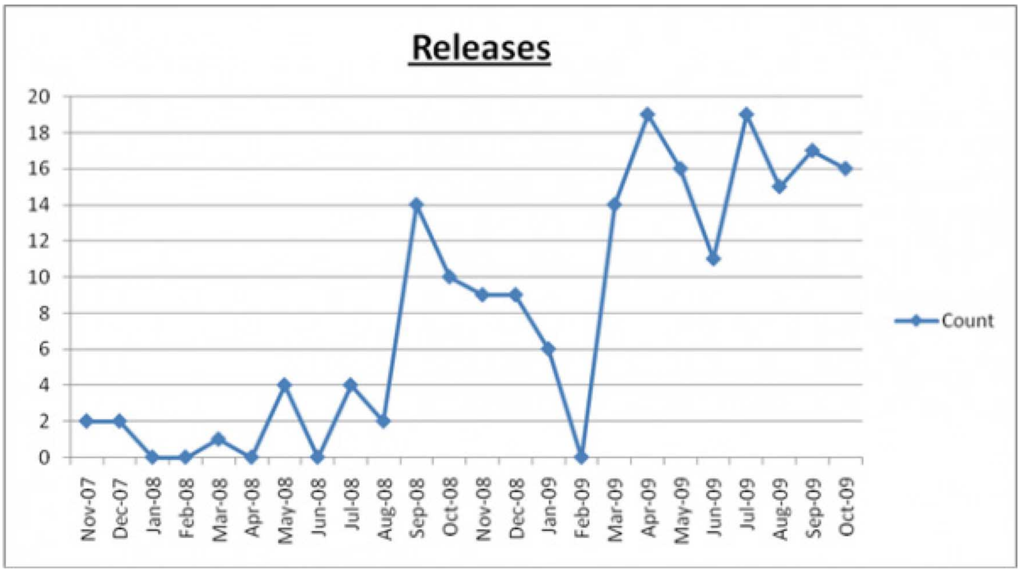

Fig. 8. Release frequency per month.

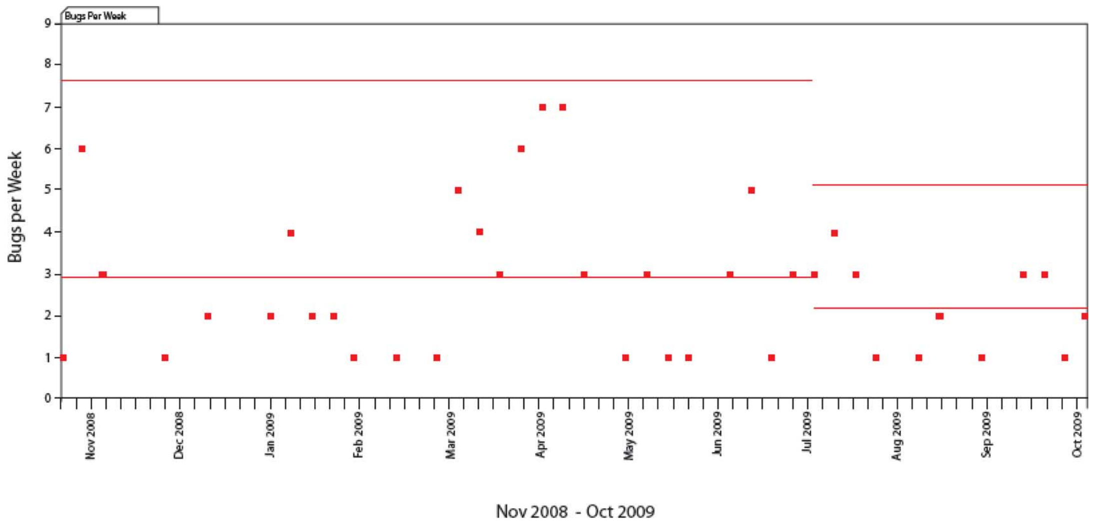

Fig. 9. Live defects per week: variance and average.

and "pull" approach meant that the size, complexity, and volume of work input were all materially reduced.

An integral part of any lean manufacturing implementation is a *variability reduction* effort, to enable a process to achieve the same (or greater) throughput with less WIP. Little's Law shows that the way to make a process more productive is to ensure the amount of WIP is not above the constraints of the system, and to also focus on reducing cycle time. Little's Law is a basic manufacturing principle that states there is a fundamental long-term relationship between throughput (output), WIP, and cycle time of a production system in a steady state [32]

$$\text{Throughput} = \text{WIP}/\text{cycletime}.$$

Throughput is defined as average output; WIP is the inventory between the start and finish points of a production process; cycle time is the time a unit spends as WIP. This equation shows that to drive productivity, cycle time must be continually reduced while ensuring WIP does not exceed the capacity available to process it.

Given the nature of software, the precise unit size and the value of the output cannot be measured exactly, but Little's Law appears to be working. Using the kanban boards to only "pull" work into the system when there was capacity to work on it ensured WIP was always at the optimum level. Cycle time was constantly improved by reducing variance through the daily continual improvement activities and minimizing the size of the units of work. The resulting rapid increase in the number of these smaller deliverables can be seen in Fig 8.

The team believed greater value was being delivered for the following reasons.

1) Only the work of highest value to customers was being processed.
2) This work was being delivered and deployed quickly so delivering value sooner.

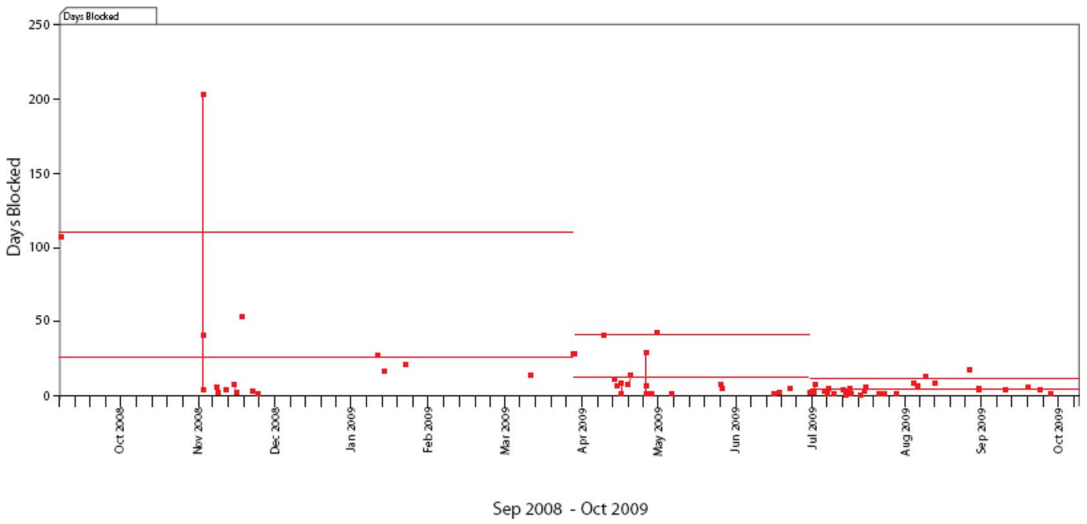

Fig. 10. Continuous improvement: issues identified and time to resolve.

3) The risk of waste by working on misunderstood or incorrect requirements was minimized.
4) Customers were reported to be happier and prefer this approach.

Software was produced more consistently with variability of delivery of lead time reduced by 47% from 70.7 to 37.3 working days over the year (see Fig. 6). This graph also shows that the mean time to deliver software features was reduced by 37% or 8.4 days from 22.8 to 14.4 working days. This means the team was more able to respond to the needs of the business by delivering new functionality faster and with more predictability.

The development time data shows even better improvement. The variance of development time fell by 78% from 30.5 days to 6.8 days (see Fig. 7). The mean development time was reduced 73% from 9.2 to 2.5 working days over the nine months. The bigger improvement in development time over lead time is felt to be because this part of the process was purely under the team's control.

The release frequency chart (see Fig. 8) does not show how much value is being delivered, but it does show an eight-fold increase in releases from 2 in November 2007 to 16 in October 2009. This indicates an improvement in configuration management discipline and capability. The more frequent releases reduce both technical and market risk by allowing customers to evaluate tangible product rather than just progress reports.

The measure of bugs opened each week, which includes those bugs not closed from previous weeks, showed improvement (see Fig. 9). The number of bugs reported by customers was low; therefore, data collected over a longer period of time would be preferred to confirm the trends. Variance fell by 33% as the upper control limit reduced from 7.6 to 5.1 open bugs per week. Defects still open and reported each week fell by 24% from 2.9 to 2.2.

This data (see Fig. 9) indicates bugs were fewer and being fixed more quickly, possibly due to the improving structure of the code base. The necessity of allowing software developers time to improve the quality of their code was mentioned by the team, as a factor in the improved bug rates. As legacy issues (technical stories) were resolved, the bug rate had fallen, thus, allowing more customer stories to be completed. Therefore, while the team was customer-focused and responsive to customer needs, they needed to pay down any technical debt to increase their productivity levels.

The data on continuous improvement (see Fig. 10) shows the variance in time taken to resolve issues shortened significantly. Over the 12 months, the mean number of working days items were blocked was reduced by 81% from a mean of 25.8 days to 4.9 days per month. The outlier in 2008 was a result of waiting for a third party to complete their work (a special cause). This does illustrate that actively looking for and recording problems increased the number of "blockers" raised, which is beneficial. These were then being removed at a faster rate by the team. This data was also used in retrospectives and quarterly reviews. Reoccurring blockers were investigated and root-cause analysis was performed.

Lean systems are typically rich in data, which could enable a team to be self-organizing and initiate continual improvement. However, as Adler and Cole identify data alone is not enough, there must be "effort to constantly improve the details of the production process" [53, p. 163]. By explicitly identifying and removing "blockers" on a daily basis, the BBC Worldwide team was constantly improving their processes.

It was not just data that was important, but the adoption of a short work cycle, which meant that: "...it is easy to identify problems, define improvement opportunities, and implement improved processes" [ibid, p.164]. By restricting the work-in-progress and improving the speed of flow through their processes, the context of a shorter cycle time enabled changes to be made.

Finally, Adler and Cole point out that by using data to focus on the progress of the work, rather than the performance of individuals, "...the knowledge required to make improvements could be used ... by the joint efforts of workers, managers, and engineers to fuel a continuous improvement..." [ibid, p.168].

It is, therefore, not only the data generated that is important but who is allowed to see and act on it. The software team studied was all able to see and act on their data to improve their work on a daily basis.

The main use of the kanban boards was to control the level of work in process and enable bottlenecks to be quickly identified. Customers rarely visited the kanban boards and did not attend the daily stand-up meetings, as they both contained more detail than customers needed and would be time-consuming. Also, the team would be working for several customers; therefore, much of the meeting would not be of interest. Customers were encouraged to attend, but overall, it was not feasible for them to be closely involved with the kanban boards.

Managers can see the status of projects on the kanban boards, but this was not the boards' primary function. Managers may ask questions when the kanban board shows several red notes, denoting live defects, or if excessive "waste" was being recorded, but otherwise they would not be closely involved with the kanban board. The regular meetings for the Steering Group or Project Committee would not be held at the kanban boards.

The boards evolved with changes to layout being made roughly on a weekly basis. They were a living tool reflecting both the evolving projects and the peoples increasing understanding of their work. The social value of the daily standup also cannot be underestimated. The requirement to daily share the progress of your work with your peers was a powerful motivator and source of discipline.

The lean approach can handle big, complex projects. The constraint is the ability of the human mind to handle complexity. Therefore, any large project can be broken up into smaller projects. A master kanban board can then be used to record and summarize the progress of all the smaller projects. There is no need or benefit from one gigantic kanban board recording everything. Scale or complexity of projects was not observed to be a problem.

Lean requires a stable, experienced team with low staff turnover and a project manager who knows the skills and abilities of their team. Given the wide variance in the talents of software developers and the people-intensive nature of software development, this is vital. Lean is not a substitute for professional software engineering practice. Effective tools for source control, bug tracking, testing, release, and deployment, were all critical enablers of the lean software process.

Using short-cycle times and the kanban boards to reduce WIP did accelerate the software development team's learning. This enabled a rapid rise in the maturity level of their development process. The learning curve was increased by the effect of Little's Law [32], which facilitates learning by shorter cycle times and the transparency resulting from reduced WIP.

One area that could possibly be improved is the "fuzzy front end" [54] by adopting "front-end loading" [17]. This refers to the time from when an idea is first discussed to when work is started on it. The time taken to move on kanban board A from "proposed ideas" to "decomposed engineering ready" is currently not recorded, and this time could be significant. Lead time could also potentially be further reduced, as customers were batching work for user acceptance testing (UAT), which was slowing the process before release ready. An investigation of how to make UAT easier for customers could be beneficial.

The concept of a fixed "heart beat" of production or Takt time used in lean manufacturing to ensure that the rate of production meets the rate of demand was used by analogy. Small chunks of work were continually released, but not in a precise, regimented way. Takt time can work more literally to pace the operation in repetitive service transactions, where the variation in the time taken for each operation is relatively low. With software, each unit of work is different as are the abilities of each software engineer; therefore, unit development times can vary widely. It would, therefore, be difficult to schedule software production using Takt time [36].

The solution was the use of MMFs [39], which could vary considerably in size and complexity. This created small definable chunks of work that reduced work in process and increased the frequency of releases allowing short-feedback loops. This provided a rhythm to production and ensured a direct connection in small high-value deliverables with the customer. When this is combined with the daily stand-ups, the work effort can be directed and controlled as the customer requires.

The normal types of obstacle to organizational change were observed. The tension with the existing standards and processes is why lean is about culture change rather than simply implementing tools. Specifically:

1) while capital costs were trivial, there is a need for considerable space to display the kanban and information boards. Organizations with offices designed around a corporate "look" may not welcome walls of post-it notes;
2) if the organization has a heavy plan driven process with standardized corporate reporting on projects, then this emergent approach will not fit easily. Lean handles risk by being highly transparent, reducing WIP, breaking projects into small parts, and frequent deliveries. Lean does not work well with targets, milestones, Gantt charts and traffic light reporting methods;
3) to truly deliver value to customers will require the development team to proactively move upstream to work with customers to define and analyze their problems and then work downstream after release to see if business value was actually created. Organizations may feel the IT teams are going beyond their remit;
4) A self-managing team can be challenging as managers need to move toward a facilitating role, which they may feel uncomfortable with. Staff may not be used to being encouraged to identify problems or having to multiskill.

VIII. Agile Versus Lean

The dearth of data in the Agile community generally is remarkable. A major study seeking evidence to support Agile reported "the strength of evidence is very low, which makes it difficult to offer specific advice to industry" [55, p. 853]. This BBC Worldwide case study may appear to record an Agile-/Scrum-like approach, where care has been taken to collect data. However, the lean approach described here does have significant differences from Agile.

A. Push Versus Pull

Scrum has time-boxed iterations or sprints with a fixed release cadence [56]–[58]. It is, therefore, in essence still a push, batch model. This lean software team lean used WIP limits to ensure a team was not overloaded. Work was "pulled" in when the team had capacity. The team did not sign up for arbitrary deadlines, as support issues could occur that could blow things off course. Arbitrary deadlines were avoided, as they tend to lead to game playing and poor quality, as attempts are made to shoehorn work into the reduced time [51].

B. Reliance On Data

Scrum has "inspect and adapt" in their retrospectives, which is a trailing indicator [59]. The boards are not so important with Scrum, because the focus is more on the people rather than the work. The Scrum "stand-up" directs attention to the people and what they did yesterday and what they are doing today [56].

In contrast, the lean team studied here enumerated the work, not the people. In the lean approach adopted data was seen primarily as a source of empowerment for the team, not as a control tool for management. The team was expected to collect and analyze their own data; so they could control and improve their own work. This lean approach used the kanban boards to expose problems and expected the team to take action. The stand-ups used the kanban boards to provide leading indicators of issues to be addressed. The lean team's "standup" focused on their work and what the team was going to release. The data was used to help the team look up and down stream to enable innovation.

C. Continual Improvement

Scrum uses "retrospectives" [60], [61], but benefits from these are largely anecdotal and not quantifiable. The concept of "velocity" measured as number of feature/story points delivered per iteration is often used. However, there is a risk that velocity estimates, number of features, or story points delivered are too subjective and easy to manipulate. Agile teams do not put the number of feature/story points delivered under statistical process control. The power of this technique to identify trends and variation in results is well established [51], [52].

In contrast, the lean team studies used "lead time," which is much harder to game, as it records total time from when a customer requested the work to when the finished work was received by the customer. They looked at "blockers" or impediments as first-class items to be addressed on a daily basis, and this drove their continual improvement activities. The lean team actively sought out data they could use for self-management and to make process improvement explicitly part of their routine.

D. Multiskilling/Collaboration

With Agile, the scrum master has the "impediment list" or "improvement backlog" [59], [60], but the responsibility for working on it can be diffuse. With this lean team because of the WIP limits and the visibility due to the kanban boards, the staff could not "cherry pick" what they would like to work on, if they were blocked. They all had to help with the bottlenecks and items blocking the work. The focus of the daily "stand-ups" was on the flow of work and not on the individual reports and performances. All staff members, regardless of their skill profiles were expected to help eliminate the bottlenecks. The objective was to deliver value as quickly as possible to the customer.

IX. Conclusion

The research hypothesis was that the application of lean ideas would improve the capability of a software development process. This would be measured in terms of reductions in lead time, error rates, and variability, combined with evidence of continuous improvement. Considering all the quantitative and qualitative data collected, the research hypothesis was supported by this single case study.

The way the work was handled did change significantly. The volume of work that was allowed to enter the process was significantly reduced to ensure the workload was not beyond the capacity of the system. The reduced cycle time meant the customer quickly received high value, small incremental deliverables. This reduced both technical and market risk.

Continuous improvement was carried out by the team on a daily basis and this may well account for the increased predictability in delivery observed. Statistical process control was seen to work well and provide useful data on trends and variance. Lean also provided a framework that encouraged other beneficial improvements, such as rewriting parts of the legacy code, developing team skills, and reducing staff turnover.

Actual business value delivered was largely influenced by the areas selected to be worked on. These strategic priorities were decided by the Business and Project Boards, not the software team. However, it is likely that as the lean software development process reduced risk, was faster, and more consistent, then greater value was being delivered to the business.

Acknowledgment

The authors exceptionally value the contribution of Prof. Jeffrey Liker and the anonymous reviewers in improving and developing this paper.

References

[1] J. K. Liker, *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer*. New York: McGraw-Hill, 2004.
[2] J. P. Womack and D. T. Jones, *Lean Thinking*. London: Touchstone Books, 1997.
[3] S. Shingo, *A Study of the Toyota Production System*. Portland, Oregon: Productivity Press, 1981.
[4] (2010). BBC Worldwide website [Online]. Available: www.bbcworldwide.com/about-us.aspx.
[5] T. Ohno, *Toyota Production System: Beyond Large-Scale Production*. Portland, Oregon: Productivity Press, 1988.
[6] J. P. Womack, D. T. Jones, and D. Roos, *The Machine that Changed the World*. New York: Rawson Associates, 1990.
[7] R. J. Schonberger, *Japanese Manufacturing Techniques*. New York: Free Press, 1982.
[8] M. Imai, *Kaizen, the Key to Japan's Competitive Success*. New York: McGraw-Hill, 1986.

[9] M. Rother and J. Shook, *Leaning to See: Value Stream Mapping to Add Value and Eliminate Muda*. Cambridge, MA: The Lean Enterprise Institute, 1999.
[10] J. K. Liker and M. Hoseus, *Toyota Culture: the Heart and Soul of the Toyota Way*. New York: McGraw-Hill, 2008.
[11] J. K. Liker and Ed, *Becoming Lean: Inside Stories of U. S. Manufacturers*. Portland, Oregon: Productivity Press, 1998.
[12] J. Drew, B. McCallum, and S. Roggenhofer, *Journey to Lean: Making Operational Change Stick*. Basingstoke, U.K.: Palgrave MacMillan, 2004.
[13] J. Allen, C. Robinson, and D. Stewart, Eds., *Lean Manufacturing: A Plant Floor Guide*. Michigan: Society of Manufacturing Engineers, 2001.
[14] G. Conner, *Lean Manufacturing for the Small Shop*. Michigan: Society of Manufacturing Engineers, 2001.
[15] M. Baudin, *Lean Assembly: The Nuts and Bolts of Making Assembly Operations Flow*. New York: Productivity Press, 2002.
[16] M. N. Kennedy, *Product Development for the Lean Enterprise: Why Toyota's Systems is Four Times More Productive and How You Can Implement it*. Virginia: Oaklea Press, 2003.
[17] J. M. Morgan and J. K. Liker, *The Toyota Product Development System: Integrating People, Process and Technology*. New York: Productivity Press, 2006.
[18] A. Ward, J. K. Liker, J. J. Cristiano, and D. K. SobekII, "The second toyota paradox: How delaying decisions can make better cars faster," *Sloan Manage. Rev.*, vol. 36, no. 3, pp. 43–61, 1995.
[19] D. K. Sobek II, A. C. Ward, and J. K. Liker, "Toyota's principles of set-based concurrent engineering," *Sloan Manage. Rev.*, vol. 40, no. 2, pp. 67–83, 1999.
[20] D. Kirkpatrick, *The Facebook Effect: The Inside Story of the Company that is Connecting the World*. Chatham, England: Virgin Books, 2010.
[21] R. Kaplinsky, *Easternisation: The Spread of Japanese Management Techniques to Developing Nations*. Ilford, U.K.: Frank Cass & Co., 1994.
[22] P. Stewart, K. Murphy, A. Danford, T. Richardson, M. Richardson, and V. Wass, *We Sell Our Time no More: Workers' Struggles Against Lean Production in the British Car Industry*. London, U.K.: Pluto Press, 2009.
[23] E. Murman, T. Allen, K. Bozdogan, J. Cutcher-Gershenfeld, H. McManus, D. Nightingale, E. Rebentisch, T. Shields, F. Stahl, M. Walton, J. Warmkessel, S. Weiss, and S. Widnall, *Lean Enterprise Value: Insight's from MIT's Lean Aerospace Initiative*. Basingstoke, U.K.: Palgrave, 2002.
[24] M. L. George, *Lean Six Sigma for Service*. New York: McGraw-Hill, 2003.
[25] B. Price and D. Jaffe, *The Best Service Is No Service*. San Francisco, CA: Jossey-Bass, 2008.
[26] P. Middleton, "Just-in-time software development," in *Proc. 2nd Int. Conf. Achieving Softw. Quality Softw., Consorzio Quality I.E.I.-CNR*, Venice, Italy, Oct. 18–20,1993, pp. 49–56.
[27] J. Tierney, "Eradicating mistakes from your software process through Poka Yoke," in *Proc. 6th Int. Softw. Quality Week, Softw. Res. Inst.*, San Francisco, CA, 1993, pp. 300–307.
[28] A. C. Hou, "Toward lean hardware/software system development: An evaluation of selected complex electronic system development methodologies," Lean Aircraft Initiative, Center for Technology, Policy and Industrial Development, Massachusetts Inst. Technol., Cambridge, MA, Report–Lean 95–01, 1995.
[29] T. Morgan, "Lean manufacturing techniques applied to software development," M.Sc. thesis, Massachusetts Inst. Technol., Cambridge, MA, 1998.
[30] T. Hamilton, "A lean software engineering system for the department of defense," M.Sc. thesis, Massachusetts Inst. Technol., Cambridge, MA, 1999.
[31] T. Gilb, *Principles of Software Engineering Management*. Wokingham, U.K.: Addison-Wesley, 1988.
[32] W. J. Hopp and M. L. Spearman, *Factory Physics*. New York: McGraw-Hill, 2001.
[33] A. Cockburn, *Agile Software Development*. Boston, MA: Addison-Wesley, 2002.
[34] M. B. Chrissis, M. Konrad, and S. Shrum, *CMMI Guidelines for Process Integration and Product Improvement*. Boston, MA: Addison-Wesley, 2004.
[35] P. S. Adler, "The evolving object of software development," *Organisation*, vol. 12, no. 3, pp. 401–435, 2005.
[36] P. Middleton, A. Flaxel, and A. Cookson, *Lean Software Management Case study: Timberline Inc.*. Lecture Notes in Computer Science, New York: Springer-Verlag, 2005.
[37] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Agile Toolkit*. Boston, MA: Addison-Wesley, 2003.
[38] P. Middleton and J. Sutton, *Lean Software Strategies*. New York: Productivity Press, 2005.
[39] D. Anderson, *Agile Management for Software Engineering –Applying the Theory of Constraints for Business Results*. Englewood Cliffs, NJ: Prentice-Hall, 2003.
[40] C. Ladas, *Scrumban: Essays on Kanban Systems for Lean Software Development*. Seattle, WA: Modus Cooperandi Press, 2008.
[41] J. Seddon, *Freedom from Command & Control*. Buckingham, U.K.: Vanguard Education, 2005.
[42] E. Willeke, D. Anderson, and E. Landes, *Proceedings of Lean & Kanban Software Conference, Miami*, Bloomington, Indiana: Wordclay, 2009.
[43] A. Shalloway, G. Beaver, and J. Trott, *Lean-Agile Software Development: Achieving Enterprise Agility*. Boston, MA: Pearson Education, 2010.
[44] K. M. Eisenhardt and M. E. Graebner, "Theory building from cases: Opportunities and challenges," *Acad. Manage. J.*, vol. 50, no. 1, pp. 25–32, 2007.
[45] A. D. Jankowicz, *A.D. Business Research Projects*. London, U.K.: Chapman & Hall, 1991.
[46] G. Walsham, "The emergence of interpretivism in is research," *Inf. Syst. Res.*, vol. 6, no. 4, pp. 376–394, 1995.
[47] J. L. Wynekoop and N. L. Russo, "Studying system development methodologies: An examination of research results," *Inf. Syst. J.*, vol. 7, no. 1, pp. 47–65, 1997.
[48] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Trans. Softw. Eng.*, vol. 25, no. 4, pp. 557–572, Jul./Aug. 1999.
[49] S. Easterbrook, J. Singer, M. Storey, and D. Damian, "Selecting empirical methods for software engineering research," in *Guide to Advanced Empirical Software Engineering*, F. Shull, J. Singer, and J. Sjoberg, Eds. London, U.K.: Springer-Verlag, 2008.
[50] J. K. Liker and D. P. Meier, *Toyota Talent: Developing Your People the Toyota Way*. New York: McGraw-Hill, 2007.
[51] W. E. Deming, *Out of the Crisis*. Cambridge, MA: MIT Press, 2000.
[52] D. J. Wheeler, *Understanding Variation: The Key to Managing Chaos*. Knoxville, TN: SPC Press, 1993.
[53] P. S. Adler and R. E. Cole, "Designed for learning: A tale of two auto plants," *Sloan Manage. Rev.*, vol. 34, no. 3, pp. 157–177, 1993.
[54] P. G. Smith and D. G. Reinertsen, *Developing Products in Half the Time*. New York: Wiley, 1998.
[55] T. Dyba and T. Dingsoyr, "Empirical studies of agile software development: a systematic review," *Inf. Softw. Technol.*, vol. 50, pp. 833–859, 2008.
[56] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Englewood Cliffs, NJ: Prentice-Hall, 2002.
[57] B. Boehm and R. Turner, *Balancing Agility and Discipline*. Boston, MA: Addison-Wesley, 2004.
[58] T. Stober and U. Hansmann, *Agile Software Development: Best Practices for Large Software Development Projects*. Berlin, Germany: Springer-Verlag, 2010.
[59] M. Cohn, *Succeeding With Agile: Software Development Using Scrum*. Boston, MA: Addison-Wesley, 2010.
[60] K. Tate, *Sustainable Software Development: An Agile Perspective*. New Jersey: Addison-Wesley, 2006.
[61] J. Shore and S. Warden, *The Art of Agile Development*. Sebastopol, CA: O'Reilly, 2008.

Peter Middleton received the M.B.A. degree from the University of Ulster, Northern Ireland, in 1987, and the Ph.D. degree in software engineering from Imperial College, London, U.K., in 1998.

He is currently a Senior Lecturer in computer science at Queen's University Belfast, Northern Ireland. He is the coauthor of the book *Lean Software Strategies* published in 2005, and the Editor of a book of case studies on applied systems thinking: the *Delivering Public Services that Work* published in 2010. His research interests include combining systems thinking with lean software development to help organizations significantly improve their performance.

David Joyce is a Systems Thinker and Agile practitioner with 20 years software development experience of which 12 years is technical team management and coaching experience. In recent years, David has led both onshore and offshore teams and successfully led an internet video start-up from inception to launch. More recently David has coached teams on Lean, Kanban and Systems Thinking at BBC Worldwide in the U.K. He is a Principal Consultant at ThoughtWorks.

Mr. Joyce was awarded the Lean SSC Brickell Key award for outstanding achievement and leadership.