Software Quality Journal, 9, 241–252, 2001

Lean Software Development: Two Case Studies

PETER MIDDLETON p.middleton@qub.ac.uk

School of Computer Science, The Queen's University of Belfast, BT7 1NN Belfast, United Kingdom

Abstract. This paper shows how the concepts of lean manufacturing can be successfully transferred from the manufacture of cars and electrical goods to software development. The key lean concept is to minimize work in progress, so quickly forcing any production problems into the open. Production is then halted to allow each problem with the system producing the goods, to be permanently corrected. While frustrating at first, the end result is very high levels of productivity and quality.

Large industrial companies are beginning to transfer their lean production expertise to their in-house software development projects. The two case studies reported here confirm that lean software development can produce rapid quality and productivity gains. A major implementation issue is that lean software development may require deep changes in the way an organization is managed.

Keywords: lean software development, just-in-time, quality, zero-defects, mistake-proofing, productivity, management, organizational change

Introduction

Lean thinking is important because it can reduce defect rates to 1 per million units (Schonberger, 1986, p. 221). It has been shown beyond question to at least double the productivity of both manufacturing and service operations (Womack et al., 1990; Ohno, 1988). It also significantly reduces the time taken to deliver new products while substantially reducing cost (Womack and Jones, 1994; Abegglen and Stalk, 1985). The evidence from all over the world: Toyota (Japan), Porsche (Germany) and Pratt & Whitney (USA), shows that lean techniques produce significant levels of improvement (Womack and Jones, 1997).

> Perhaps the most striking difference between mass production and lean production lies in their ultimate objectives. Mass producers set a limited goal for themselves—"good enough," which translates into an acceptable level of defects, a maximum level of inventories, a narrow range of standardized products. To do better, they argue, would cost too much or exceed inherent human capabilities. Lean producers, on the other hand, set their sights explicitly on perfection: continually declining costs, zero defects, zero inventories, and endless product variety. (Womack and Jones, 1997)

The vital concept for software developers is that instead, for example, of producing a large requirements documents and then passing it over to design to work on; instead as each page is finished it should be passed onto the designers and developers. But as soon as they find an error or inconsistency in this one page of

requirements it must be returned to the analysts for correction. In this way the analysts receive immediate and relentless feedback on their performance. Recording and analysing returns to the analysts clearly highlights any organizational problems concerning their selection, retention, motivation and training.

Lean software development can be seen as a refinement of Fagan's (1976) seminal paper, which showed how to apply statistical quality and process control methods to "ideas on paper." It also combines Gilb's (1988) evolutionary software development approach with the insights from the Toyota car production system (Ohno, 1988).

The paradigm shift required for software is to see that the paper specifications and unfinished programs are in effect stocks or work-in-process. The real cost is not the storage of the paper; the real cost is in the errors that remain undiscovered in the documentation until later. By that time the authors have forgotten all the rich details of the situation they were describing and the project has moved on; so correcting the errors will be far more expensive. There is no immediate feedback to catch the errors and correct the process that created the problem in the first place. Therefore the analyst who, through faulty technique or carelessness, introduced the errors will go on producing them elsewhere. This is the heart of lean techniques applied to software development.

The key flaw in lean production is clearly described by Pascale (1990) and confirmed by these case studies. That is, to adopt lean concepts successfully the entire organization has to change. This is particularly hard for senior staff who are by definition, successful under the existing system. As the data from Collins and Porras (1998) shows, successful companies have a clear alignment of what they stand for, the type of people who thrive in them and their internal systems. They all fit together. Lean practices are not hindered by nationality, but the scale of organization change they require can severely limit their adoption.

Robinson and Stern (1998) in their study of corporate creativity, show that it is essential that there is alignment so that the "... the interests and actions of all employees are directed toward a company's key goals, so any employee will recognize and respond positively to a potentially useful idea" (p. 13).

Essentially, the clarity of purpose and employee alignment required for lean techniques to work is very difficult to achieve. If there is a willingness to change then lean techniques can guide and accelerate the changes required by rapidly unveiling organizational barriers.

Literature review

Schulmeyer (1990) of the Westinghouse Corporation reports on the application of ideas for achieving zero defects from the manufacturing industry to software development. His focus is very much on defect prevention, which is carried out by the extensive use of checklists, mistake proofing, metrics collection and inspections. In this early work he estimates that savings of up to 40% are possible with this approach.

Tierney (1993), a Quality Assurance Manager with Microsoft, describes a successful application of Japanese quality assurance theory to software engineering. The

particular focus is to develop tools and processes that eliminate errors as close to the source of the error as possible. The relationship focused on is that between the programmer and the tester.

Tierney's experience, confirms Boehm's (1981) data, that the further along in the software process a mistake is found, the more expensive it is to correct it, often by orders of magnitude. Poke Yoke (mistake proofing) is the systematic practice of eradicating whole classes of errors from the software development process by locating the root cause of the errors and eliminating the potential for making that mistake.

In lean manufacturing it is essential to reduce inventory because mistakes caused by a defective process can hide in the inventory (more units ruined before the mistake is detected). In software development the same principle applies: software inventory (requirements, design, code) hide mistakes which can surprise you and ruin your schedule if you do not fix them early. The key point is "reducing inventory = reducing untested code" (Tierney, 1993).

Hou's 1995 report 'Towards Lean Hardware/Software System Development' evaluates the latest system development methodologies for the US Department of Defense. Hou's conclusion is that the only way forward is with lean techniques.

Lockheed Martin Aeronautical Systems reported they had started using lean software development in 1986 (Sutton, 1996). Based on this 10 years of experience Sutton concluded ". . . the lean paradigm is enabling the software organization to produce a safe and superior product at low cost and risk. . . . What more could one ask from a development paradigm? And how long can the older paradigms last in a world where these kinds of results are available?"

Morgan (1998) from Cummins Engine Company: ". . . provides some compelling evidence that the ideas of lean manufacturing are indeed applicable, in principle, to software development. The data will show that lean software development exists and can be practiced." Morgan's work is important because it takes the most common type of software process: 'While some characterised the software development process at Cummins as non-existent, it was a typical first-generation process.'

Morgan (op cit.) states "Clearly implementing, even badly, the lean software ideas from the case result in dramatic improvements." This is because the naturally occurring schedule pressure in the project drives the improvement loop very strongly. The strong improvement loop drives up productivity.

A study for Boeing carried by Raman (1998) 'Lean Software Development: Is it feasible?' concludes: 'Lean thinking is applicable to any activity or processes including software development processes.' This work has no empirical data but looks at emerging software development methods and compares them to the application of lean principles.

Hamilton (1999) in a study 'A Lean Software Engineering System for the Department of Defense' concludes that ". . . shifting to lean principles improves cycle time reduction and overall quality in the software development process."

Applying lean approaches to software process follows from the current belief that it is the maturity of the software process that determines success (Paulk et al., 1993). Once software process is seen as a systematic process, then the techniques

and experience from the other industries, which have long recognized a production process, can be used. The difficulties of implementing and sustaining a mature software process are clearly recognized by Humphrey (1998).

The essence of lean software is 'stockless production,' the production of small quantities 'just in time.' The ideal is for all requirements, designs and code, to be in active use as elements of work in process, never at rest. It is a hand-to-mouth mode of operation, constantly striving towards smaller units of work. It is like perfect quality—never attained, but rather an ideal to be pursued aggressively (Schonberger, 1982)

To supplement this core idea of 'stockless production' are five familiar principles of quality improvement (Schonberger, 1986):

- continual quality improvement, project after project;
- empowered workers taking responsibility;
- defect prevention, not random detection;
- simple, visual measures of quality;
- automatic quality measurement devices, often self-developed.

Contrasting ISO 9000 with lean software development highlights the fundamental differences. Seddon (1997) observes that: "ISO 9000 starts from the flawed prescription that work is best controlled by specifying and controlling procedures." (p. 178). This requirement to "Document what you do and prove that you do it" rapidly causes an organization to sub-optimise. This explicitly discourages organizational learning. This is opposite to lean software development, which uses simple, visual measures of quality to drive organizational learning.

The key problem with inspections as Gilb and Graham (1993) make clear, is that after 25 years their success is still constantly hindered by many technical, organizational and implementation issues. The critical factor preventing organizations from accepting inspections as a technique is that they will require major changes within the organization. It is simply too much to ask for a technique presented as a technical solution to a software quality problem, to drive significant organizational change. The key insight from the long experience of lean techniques in other industries is that quality simply cannot be bolted onto a fundamentally misaligned organization. The potential power of inspection is undoubted, but they do not and cannot go far enough to be implemented successfully. The key limitation is that inspection will hit barriers to its effectiveness, which are outside its self-described terms of reference.

A culture such as General Motors summarized by one employee as "Let's face it—at GM we are literally applauded for being able to cover up problems" (Pascale, 1990, p. 244). This type of organization is going to have problems with any approaches based on openness.

The lean approach is holistic and can encompass the software process. Lean techniques work by rapidly identifying why and how a person is making mistakes. They then provide the data to assist the necessary changes to the organizational infrastructure. This rapid organization learning and alignment of people, process and goals enables people to produce the quality of work required. By starting with the acknowledgement that an organization is misaligned quality improvements can

rapidly take place. Lean production has always presented itself as a complete management system dedicated to the elimination of waste.

Research methodology

The need was to find a way of experimenting with lean techniques applied to software development. Two main options were identified.

One was to construct a laboratory experiment using computer programmers on a simulated project. This would have been more in line with the classic research model, but was rejected as being too difficult to arrange and could not replicate the full detail of commercial projects.

The other was to use the interdisciplinary applied participatory action research model described by Whyte (1991). Participatory action research is defined as involving: 'practitioners in the research process from the initial design of the project through data gathering and analysis to final conclusions and actions arising out of the research.'

Cunningham (1993) identifies the difficulties involved in using the traditional scientific research paradigm for practical organizational development problems. He argues that: "Action research is a unique scientific process in its concerns for accepting the values inherent in the client system and differs from the traditional scientific inquiry because of the researcher's involvement with peoples' expectations and values."

Lee and Baskerville (2001) make the important point that if a study is not statistical, then it is not valid to use a statistical metaphor to say that we can't generalise from the study. This paper is generalising from a theory to a new setting. It is concerned with the applicability of lean techniques to software development. The data and literature provided do not disprove this theory. The limitation is that the data currently available indicates some validity to the theory but is not rich enough to be conclusive.

The criticisms of the approach taken are that it lacks objectivity and is not easy to replicate. In practice there is no reason that a similar experiment could not be created by other researchers. There is enough detail provided for the experiment to be repeated. The choice here was to use the participatory action model because it was not possible to apply the classical model on a project of any meaningful size or duration.

Empirical work

To test the applicability of lean techniques to developing software, the following participatory action project was instigated and carried out. An organization employing 7000 people was approached; its Information Systems Department agreed to allocate enough personnel for two teams. Each team consisted of an analyst, programmer/analyst and a programmer. The two teams, A and B, used for this lean

experiment both shared the same large office and used equipment of the same specification. The reason for this experiment design was to neutralise any Hawthorn-type motivational effects of any differences caused by accommodation or facilities.

The two teams were writing enhancements and corrections for the same financial and management information system which was in constant use. The system was made up from software packages that had been heavily modified by outside contractors. Technical documentation was virtually non-existent. The system's programs consisted of 200,000 lines written in the Structured Basic computer language. The listings contained few comments and several of those present were found to be inaccurate. There was no consistent naming convention for variables. Many people had worked on these programs, so the styles and standards used in different parts of the code varied considerably. The programming staff were demoralized because of the frustrations of working with this code. Projects ran consistently late by large margins. Even small projects took weeks to complete.

This site was considered to be a good one to test the concepts of lean production because it was typical of the majority of software organizations (Humphrey, 1998). The financial system paid out supplier cheques of on average £2.7 (US$4.3) million per week, and the staff were afraid that this process would become faulty. The system also generated weekly finance and management reports to the Board, and again there was the risk that these would be wrong. The organization collected no statistics on the performance or productivity of its programming staff. Time sheets were completed but these measured attendance rather than work accomplished.

Before any progress with lean production could be made, it was necessary to clean up the software development process. The staff was working hard and was skilled in the programming language and operating system. The process used by the staff was inconsistent from team to team and within each team, and the results were erratic. The process was also very difficult to manage because neither the quality nor quantity of the work produced could be determined. Work scheduling was also impossible because all projects were taking longer than anticipated. The unstructured process hindered learning in the organization. New staff members learned by trial and error, or sometimes from a more experienced staff member through informal on-the-job assistance. There were no standards or quality assurance procedures.

Therefore, before the lean techniques could begin it was necessary for the two teams to spend six weeks developing, documenting, and agreeing upon a simple set of standards and procedures for them both to use.

The clean-up changes made to the process for these two teams to use were allowed time to settle down, which they did in a few days. This also gave the teams time to learn and understand the refined process. But after four weeks it was apparent that the software that the teams produced still contained errors, and was still not being delivered when expected. It was also observed that the standards were not always being followed. While, with added experience, the standards were developing, a climate of continual improvement had not been created. It was at this point that lean working was started.

Lean working introduced

When it was clear that the agreed standards were not being followed and that the quality reviews were missing defects in the teams' work, the lean approach was initiated. Both teams selected tasks that they agreed were of comparable difficulty. The manager was no longer to have a direct role in quality assurance. The analysts were asked to pass on the work file to the programmer as soon as a page of analysis work was completed. A page was used as a guideline so that the work-in-process represented by the documentation would be minimized in order to comply with the lean paradigm. The staff were not to 'stockpile' large amounts of their work. Previously the analysts would spend days or weeks on analysis before passing the results of their work on to the programmers. By using lean techniques it was anticipated that data would be generated and recorded throughout the day on the process as errors were found.

The programmers were instructed to return immediately to the analysts any work that either did not conform to standards or that contained errors. The programmers recorded errors by tally marks on a sheet at the front of the project file. Based upon the experience of the teams, the common error categories were:

- incorrect use of naming conventions;
- illogical pseudocode;
- omissions from the project file;
- programmer unable to understand the analysis.

The programmers were also instructed not to take on any other work while the analyst was amending the file. They were allowed to assist the analyst to correct the fault if required, or to do 'housekeeping' or reading. The reason for this was to focus attention on the process error to ensure it was resolved promptly. To comply with the lean paradigm it is essential to stop work and correct process errors as they appear, to avoid the accumulation of error-ridden work-in-process.

Results of introducing lean working

Team responses

In both teams it was observed that the files were returned to the analysts either immediately on receipt because of an obvious error or after typically about half an hour as the programmers settled into the detail (see Figure 1). This was a time of frustration and low productivity. This is exactly what took place with Mr Ohno in Toyota in the 1940s as described by Womack et al. (1990): "Not surprisingly, as Ohno began to experiment with these ideas, his production team stopped all the time, and the workers became discouraged." This initial disruptive and disappointing phase gradually passed: "However, as the work teams gained experience identifying and tracing problems to their ultimate cause, the number of errors began to drop dramatically." In the present software process experiment this also happened. The

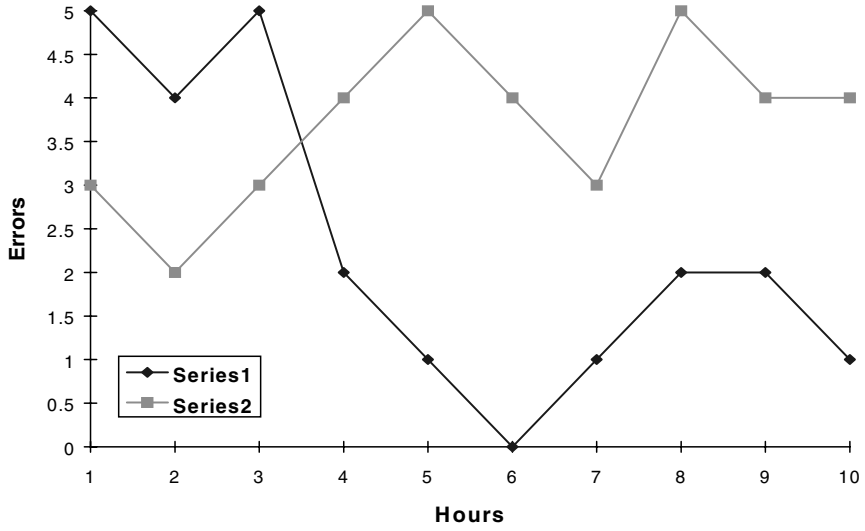

Figure 1. Pattern of errors reported.

analysts and programmer/ analysts became much more careful in their work. But even more valuable was the difference that emerged between the two teams.

Team A (Series 1). This was originally seen as the weaker team because they were younger and less experienced. None of them were graduates and they all had poor academic records. They had been selected for the Information Systems Department on the basis of their excellent performance in clerical and junior administrative positions. They quickly developed a constructive work rhythm and the standards were being continually enhanced by their suggestions.

Team B (Series 2). The analyst and the programmer/analyst were graduates in Computer Science and the programmer was studying Computer Science with employer support. The analysts were in their mid-thirties and had worked in a variety of companies. They had a reputation for being 'gurus' and were well liked by the user community for their relatively quick fixes to problems. But the lean error sheet quickly showed that the analyst in this team was chronically error-prone and had an extreme dislike of documenting anything. The situation was allowed to continue for a further two days, but the analyst was unable to produce work without errors. This situation was awkward. Because the infrastructure necessary to go to the root of this problem was not present, the experiment was stopped at this point after running for 3 days.

Post-experiment interviews

Team A. This team's better performance seemed to be due to three factors. First, they were delighted to be given the chance to work with computers and were there-

fore keen to make a success of it. Second, they were used to and expected to have their work reviewed and commented on. And third, they saw the work as a job to be done methodically and not as a series of clever tricks.

Team B. The analyst did not see his future in software process or development but in general management and felt he was over-qualified for his present position. He felt that by keeping his knowledge to himself he was protecting his job. He also felt that his freedom was restricted if people knew what he was doing. This analyst was experienced and personable, but identified with the 'black magic' or 'secret art' school of software. A review of his past projects revealed this as a deep-seated trait. In the past he had either been left alone to tackle specific small jobs, or the programmers had been working around his specifications. Discussions with his team-mates indicated that they felt they had suffered for far too long from his sloppy work.

Sustainability

It is important to note that Team A could not sustain the lean method of working for more than a few days after the experiment finished. There were four main reasons for this:

1. The organizational hierarchy within the teams of:
   Analyst, Programmer/analyst, Programmer was seen to be hindering communication. If people were expected to contribute their energies to improving the process, rather than just doing their job, then a grading based on contribution, and the removal of hierarchy-identifying titles would be needed. The seniority structure made it hard for the 'junior' member of team, the programmer, to report errors in and comment on the work of a 'senior' member, the analyst. Lean organizations reduce the job demarcation implied by the different titles and have flatter organizational structures.
2. The traditional pattern of promotion used in the organization studied, from programmer through to analyst, is not readily compatible with lean techniques. An experienced programmer can pull along and improve a weak analyst. Such programmer contribution needs to be formally recognized. The necessary skills are quite different in each role, and a good programmer may not make a good analyst, and vice versa. In the organization studied the promotion approach encouraged people to take jobs for which they had little aptitude.
3. The idea of lean thinking is to reduce the slack in a process to force sources of errors into the open. As software is a 'people intensive' business it would be reasonable to expect that 'people related' issues would surface. Reducing the slack in the software process did highlight problems in linking processes. For example, within the organization studied the human resource management was weak in selecting and retaining a person not well matched to their job. The Personnel Department that had recruited the analyst was not prepared to support his retraining, or redeployment. The Personnel Department did not really understand the significance of the errors in the analyst's work. The Department also

knew that if it got involved, this situation could require considerable amounts of its time, which it was reluctant to commit to a case based on unsatisfactory work. A pattern of internal suppliers—such as office maintence or purchasing services indirectly causing software problems by failing to deliver the quality of service required was observed. This observation is confirmed as normal by the lean literature.

4. Organizational learning and improvement was hindered by the company rules. For example, one of the programmers wished to attend a short course that was agreed to be valuable by the manager; but the company had a regulation that staff of that grade were not allowed travel expenses. It was also difficult to make small physical changes to the workspace because of the need for budgetary approvals from the Finance Department and the requirement to put all work through a Building Process Department. Both of these internal suppliers were slow to respond and awkward to deal with.

The lean paradigm meant that the programmers did, on occasions, have to do other tasks, while the errors in the analysts' work were resolved. Under lean production this is more useful than continuing to use a faulty production process. In practice this slack time was minimal because of the speed of response from the analysts. The experience of the motor industry is that as a process matures and defect causes are removed, disruptions become rare. The lean technique produced data that reversed the perception of which was the better performing team. It also provided data, which allowed probing questions to be asked on why and how the job was being performed. In this case it was clear that the talents of the Team B analyst were not being effectively employed. Because of this his errors were being seeded into the software.

Conclusion

Lean software development indicates that software quality problems are often the result of deeply embedded organizational habits of recruitment, retention and motivation. Poor software quality is usually a manifestation of deeper problems inside an organization. To obtain organizational change there is a need for fast results from low cost actions. Change requires motivation, which is triggered and sustained by results.

Software quality techniques are often overlaid on existing procedures, which means their effectiveness is inherently limited. By reducing the inventory of requirements, design and code, faults are made immediately and painfully visible. This rapidly causes reflection on the process flaw that is causing the pain. This brings into stark relief the real causes of the poor software quality.

The lean technique has demonstrated that it can go right to the core of the problems of motivation, quality assurance and staff evaluation. This is consistent with the findings for the software application and individuals studied here. The use of simple tally marks to record errors produced by the analysts is a way of implementing the lean principle of visual measures of quality control of this part of

the process. By moving responsibility for measuring quality from the manager to the workers, a much quicker and more thorough response to defects was obtained. In the present case, adding technology or prescriptive methodology to the team could not have helped the under-performing analyst. The root of the process errors was a mismatch between the analyst's career aspirations and the job he was assigned to do. The challenge thrown up by lean techniques was to find this talented and well-qualified individual a role in which he could contribute successfully, and to replace this individual with a more appropriate analyst.

Implementing lean thinking helped by providing data on how the process was performing within about 15 minutes from the start of the experiment. It highlighted defects well before they became errors. By not being prescriptive on how the work was to be done, human ingenuity was being constantly challenged to find out why the process had stopped and what could be done to correct it. The manager now had fresh clear data on how the teams were running. This was an invaluable supplement to the normal techniques relied on by the manager: observation, discussion and comments from users of the software service.

In this attempt the lean techniques revealed policy, organizational structure, communication, and internal service barriers to improving software process. This shows the challenge generated by lean thinking in practice. Womack and Jones (1994) show how even a successful lean initiative can be pulled off course by the actions of others inside a company.

From this study, no inherent reason has been found to suggest that lean techniques cannot be used in software process. With knowledge workers it is often difficult to establish exactly what stage a project has reached and who is working effectively within it. It is far more tempting for managers to invest funds to attempt to solve productivity problems by buying new software tools, than it is to take on thorny personnel issues. But it is people and how they are managed that are critical. In this experiment the lean techniques have been demonstrated to go right to the core of the problems of motivation, quality assurance and staff evaluation.

Trying to create sustainable competitive advantage by buying equipment is ineffective because this strategy is so easily copied. But a competent well-performing staff is harder to acquire quickly. Lean methods provide feedback, which motivate the staff and provide clear evidence of daily performance. Problems are identified and this then puts the responsibility on management to resolve the issues that are causing the problems. So it is concluded from this work that, in the same way as lean techniques have transformed parts of the manufacturing industries, they do have the potential to transform software development.

References

Abegglen, J.C. and Stalk, G. 1985. *Kaisha, The Japanese Corporation*, New York, Basic Books.

Boehm, B. 1981. *Software Engineering Economics*, Englewood Cliffs, New York, Prentice-Hall.

Collins, J.C. and Porras, J.I. 1998. *Built to Last: Successful Habits of Visionary Companies,* London, Century.

Cunningham, J.B. 1993. *Action Research and Organizational Development,* Westport, CT, Praeger.

Fagan, M.E. 1976. Design and code inspections to reduce errors in program development, *IBM Systems J.* 15(3): 182–211.

Gilb, T. 1988. *Principles of Software Engineering Management,* Wokingham, England, Addison-Wesley.
Gilb, T. and Graham, D. 1993. *Software Inspection*, Wokingham, England, Addison-Wesley.
Hamilton, T. 1999. A Lean Software Engineering System for the Department of Defense, Massachusetts Institute of Technology, MSc Thesis.
Hou, A.C. 1995. Toward Lean Hardware/Software System Development: Evaluation of Selected Complex Electronic System Development Methodologies, Report—Lean 95-01, Lean Aircraft Initiative, Center for Technology, Policy and Industrial Development, Massachusetts Institute of Technology. *http://lean.mit.edu/public/index.html*.
Humphrey, W.S. 1998. Why don't they practice what we preach? *Annals of Software Eng.* 6: 201–222.
Lee, A.S. and Baskerville, R.L. 2001. Generalizing Generalizability in Information Systems Research, seminar paper presented at The Queen's University of Belfast, 2 May.
Morgan, T. 1998. Lean Manufacturing Techniques Applied to Software Development, MSc Thesis, Massachusetts Institute of Technology.
Ohno, T. 1988. *Toyota Production System: Beyond Large-Scale Production,* Cambridge, MA Productivity Press.
Pascale, R.T. 1990. *Managing on the Edge,* London, Viking.
Paulk, M.C., Curtis, B., Chrissis, M.B., and Weber, C.V. 1993. Capability maturity model, version 1.1, *IEEE Software* 10(4): 18–27.
Raman, S. 1998. Lean software development: Is it feasible? *17th Digital Avionics Systems Conf.*, New York, IEEE, pp. 13–18.
Robinson, A.G. and Stern, S. 1998. *Corporate Creativity: How Innovation and Improvement Actually Happen*, San Francisco, CA, Berrett–Koehler.
Schonberger, R.J. 1986. *World Class Manufacturing*, New York, The Free Press.
Schulmeyer, G. 1990. *Zero Defect Software,* New York, McGraw-Hill.
Seddon, J. 1997. *In Pursuit of Quality: The Case Against ISO 9000,* London, Oaktree Press.
Sutton, J.M. 1996. Lean software for lean aircraft, *15th Digital Avionics Systems Conference*, New York, IEEE, pp. 49–54.
Tierney, J. 1993. Eradicating mistakes from your software process through Poka Yoke, *Proc. 6th Inte. Software Quality Week*, Software Research Institute, San Francisco, CA, May, pp. 300–307.
Whyte, William F. 1991. *Participatory Action Research*, London, Sage.
Womack, J.P. and Jones, D.T. 1997. *Lean Thinking*, London, Touchstone Books.
Womack, J.P. and Jones, D.T. 1994. From lean production to the lean enterprise, *Harvard Business Rev.* 72(2): 93–103.
Womack, J.P., Jones D.T., and Ross, D. 1990. *The Machine that Changed the World*, New York, Rawson Associates.

Dr. Peter Middleton is a Senior Lecturer in Computer Science at The Queen's University of Belfast. He joined the university after spending 10 years in industry. He specializes in software quality and is now focusing on the rapid development of e-commerce applications. He has given guest lectures on his research in Europe, America, and Asia. He has the following degrees: BA (Econ) Manchester University; MBA University of Ulster; MSc The Queen's University of Belfast; PhD Imperial College, London.