Lean Configuration Management – Supporting Increased Value Delivery from Agile Software Teams

Jens Norin
Softhouse Consulting,
Stormgatan 14,
SE-211 20 Malmö,
Sweden
jens.norin@softhouse.se

Daniel Karlström
Dept. Communication Systems,
Lund University,
Box 118, SE-221 00 Lund,
Sweden.
daniel.karlstrom@telecom.lth.se

ABSTRACT

Configuration management, as a discipline for supporting software development, has been around for half a century and has evolved into standard practice within traditional software development processes. One of the key purposes of configuration management is to control changes made to the software product.

Agile development methods are becoming increasingly popular and claim to embrace change. These methods put emphasis on responsiveness to change rather than controlling change. How can these contradicting methodologies co-exist in the one and same project? This paper investigates the integration of configuration management and agile methods by introducing the concept of Lean Configuration Management, and shows how this can support an increased return on investment of the software project.

While traditional configuration management evolves around controlling changes of a forecasted plan, lean configuration management is about handling changes in an adaptive way as well as supporting the values of agile development and lean principles.

Categories and Subject Descriptors

D.2.9 [**Management**]: Software Configuration Management.

General Terms

Management, Performance.

Keywords

Agile, Lean, CM, SCM, Configuration Management.

SERPS'06, October 18-19, 2006, Umeå, Sweden.

1. INTRODUCTION

Configuration management has evolved from the beginnings of software engineering in the 1960's as a key discipline, required to assert control over changes and handle coordination between concurrent work by individuals and teams. As configuration management has evolved from a need of control, this contradicts the values of the Agile Methodologies [13] and Lean manufacturing [4] that emphasize responding to change rather than controlling change.

The goal of this paper is to present how to take benefits from the attractive and necessary parts of traditional configuration management and apply them to agile methods and lean principles. The opposite is also presented, i.e. the waste products within traditional CM and how to replace them with more agile counterparts.

This goal is met by discussing experiences gained from CM activities in two industrial organizations over a period of more than a decade. These experiences lead us to the conclusion that by allowing the CM activities to support agile principles we can achieve several interesting effects, both as a direct result of the CM activities and the effects of the agile practices supported. Specifically, by making CM practices support agile methods and lean principles we assert that it is possible to reduce the overhead involved in CM activities, increase the possibility of rapid feedback through more frequent iteration deliveries and to be able to control more rapid change than traditional CM activities can support. The results of these effects should ultimately lead to increased ROI for the software project.

This experience paper is the result of cooperation between Lund University, Sweden, and Softhouse Consulting, Sweden. The cooperation aims to allow researchers a greater insight into industrial software development environments by gathering experiences gained by consultants.

The paper is structured as follows. In Section 2 we provide a short oversight of Software Configuration Management (SCM), in section 3 we discuss agile principles and their implications on SCM, in Section 4 we present a concept of Lean Configuration Management building on the discussion in Section 4 and experiences from the industrial cases presented. Section 5 discusses the methodology and section 6 summarize our conclusions.

2. SOFTWARE CONFIGURATION MANAGEMENT

2.1 CM Definitions

In this section a number of different CM definitions are quoted in order to illustrate their common properties and to be able to compare them with agile development ideas. Especially note that they all talk more or less of how to deal with changes.

IEEE - "Configuration Management is the process of identifying and defining the items in the system, controlling the change of these items throughout their lifecycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items." [11]

CMM - "…Software Configuration Management involves identifying the configuration of the software (i.e., selected software works products and their descriptions) at given points in time, systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the software lifecycle. The work products placed under software configuration management include the software products that are delivered to the customer (e.g., the software requirements document and the code) and the items that are identified with or required to create these software products (e.g., the compiler)…". [18]

RUP - "The task of defining and maintaining configurations and versions of artifacts. This includes baselining, version control, status control, and storage control of the artifacts." [17]

2.2 Traditional CM Practices

As we could see in the previous section several similar CM definitions exist. Below a number of different CM activities are presented that define traditional CM for the scope of this paper.

Identify Configuration Items - Identification of all configuration items, i.e. selected work products and their descriptions, including source code files, documentation etc. It also involves arranging the configuration items in a structure that reflects the structure of the product.

Version control of configuration items - Version handling of configuration items is necessary to have traceability and facilitates multiple developers working within the same code base. It is especially important for incremental releases, when we have one version or release of the product in production/maintenance and one or more in various states of development. Except for extremely small products it is necessary to have a tool that supports version handling. In its simplest form a single branch or a more advanced alternative including parallel branches with built in merging capabilities, to support the copy/merge model.

Release management – Release management is about planning and executing external releases to the customer as well as internal releases to the project. It often includes creating a release plan and a baseline plan. In traditional development methods with few releases, release management is a very formal procedure, while in agile methods it is more frequent and less formal.

Build management – Build management is about integrating and building all the parts that make up the product. A lot of efforts can be saved by scripting and automating the integration and building process with tool support. The agile community prescribes continuous integration.

Control changes - Because of the complex nature of developing software products we can in most cases expect changing requirements. Changing requirements must be handled in a controlled way. In traditional methods this is handled by a Change Control Board (CCB) that approves and plans requests for change. Both the customer and the project is represented in the CCB.

Track status – The status of ongoing development of software components and change requests of existing components are tracked and reported. Statistics based on these statuses can then be collected and reported, and could be the basis for further decisions. The agile equivalent of tracking status is the product backlog and sprint backlog.

Audit – The CM audit is an independent review of a work product to assess compliance with all specifications and standards.

3. AGILE DEVELOPMENT

3.1 CM in Agile Methods

Most agile methods do not mention any explicit CM routines or practices, simply because it is out of the context of the agile methods. E.g. XP and Scrum are more of project frameworks that do not include CM in their scope. XP does however rely heavily on working CM routines for some of its engineering practices. [1][2]

The Unified Process (UP) however describes a heavy and rigorous CM process, but in the context of this paper UP is not considered agile. [6]

3.2 Lean Principles

Since CM is a supporting discipline and not actually primarily involved in creating any executable code, the whole discipline could by lean principles be considered as waste. However since CM is supposed to facilitate the production of executable code, it is not considered to be waste. A total lack of CM would lead to a chaotic situation without any progress at all in most projects, so the difficulty lies in identifying just enough CM to be effective without waste. [4]

3.3 Agile Manifesto

The following discussion refers to the effects applying the various parts of the agile manifesto to a CM context. [13]

Individuals and interactions over processes and tools - CM is heavily focused on processes and tools, which could imply that CM is of lesser importance than the actual individuals performing them. However lean CM aims to trim the process and automate the tools, which moves focus back towards individuals again. Instead of adapting the process to the tools, which is far too common, we try to adapt the tools to the process.

Working software over comprehensive documentation - Since CM is a supporting discipline it does not say anything about the configuration items it is set up to handle, whether it is working software or comprehensive documentation.

Customer collaboration over contract negotiation - This is a relevant issue within the CM context. A CCB routine could in worst case be an endless contract negotiation to decide whether an issue is a change request (customer cost) or an error (project cost). Having some kind of an "agile contract" [21] between the customer and the project would be preferable and would replace the formal CCB with Scrum backlogs as the CCB process has many similarities. The list of change requests corresponds to the product backlog. The CCB (product owner in Scrum) is responsible for prioritizing the list and decides which change requests should go into the next release (sprint in Scrum). This implies that all decisions regarding changes, including those on the critical path [27] can be made instantly in Scrum, whereas with a traditional CCB the decision is held up until the next CCB meeting, resulting in unnecessary delays.

Responding to change over following a plan - Controlling change is one of the foundations of CM which makes CM activities very important when focusing on responding to change instead of limiting change so as not to ruin the plan.

3.4 Roles

Traditional development methods have a more developed role concept i.e. more defined and specialized roles. E.g. in RUP the CM discipline is covered by as much as three distinct roles. This is interpreted by many projects that different people need to be allocated to each role, which is a mistake leading to over populated projects and individuals that are far too specialized.

Scrum has fewer roles, and instead let the teams self-organize. However this does not mean that the CM function as such is unnecessary, on the contrary. It means that the CM functions must be performed by the team as a whole.

The agile methods imply a lot of tacit CM knowledge within the team. If that knowledge does not exist within the team it must be introduced in some way. If knowledge and experience does exist or is temporarily hired within the team the knowledge transfer can be solved by pairing. Pairing is an XP practice where two team members temporarily pairs to do tasks together and is a great way to transfer knowledge and experience [24]. E.g. a person with the necessary CM knowledge and experience is a member of the development team for a few iterations and for CM issues pairs with all the other team members in turn, in order to transfer knowledge.

3.5 Agile Practices

This section describes a number of agile practices that to various degrees can be related to traditional CM practices. And by adapting these agile practices into the CM process we get a leaner process in traditional projects and of course in agile projects.

The agile practices described in this section are mostly engineering practices from XP and project practices from Scrum.

Embrace Change - Embrace change is a common phrase within the agile community, and handling change is one of the key concepts within CM.

In traditional development there is a clear distinction between change requests, when the requirements are wrong, and defects, when the implementation of the requirements is wrong. When having a backlog where all changes are represented it becomes less important to distinct between different kinds of changes. They all have to be done, are all prioritized together and, most importantly, require the same resources to be resolved. However embrace change does not mean embrace chaos, and further more embrace change is not possible without appropriate CM routines that support and enhance the agile ideas.

Copy/merge model - The copy/merge model [15] is popular within both the agile community as well as more traditional environments. In this model each developer has a branch of their own, seeded from the main integration branch. All development takes place in the developer branches and as soon as a change is completed and tested it can be integrated onto the main integration branch. These changes must then be rebased to all the other developer branches to make sure everyone has the latest changes [Figure 1].

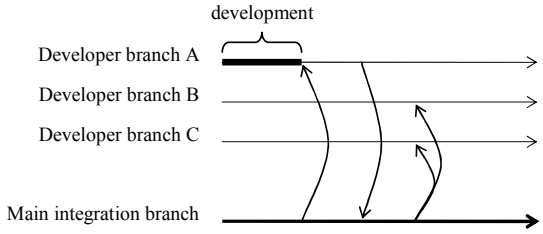

Figure 1: copy/merge integration model [15]

To avoid integration problems developers must integrate often, and as soon as anyone has integrated everyone has to sync their development track to the integration track (daily).

Continuous Integration - Whether a traditional project or an agile project frequent integration is crucial. The more frequently you integrate the easier each integration will be. With continuous integration the code is more or less always integrated and the drift from stability is on an extremely small scale. A failure simply reverts to the previous version. On the other extreme of the scale, late integration could lead to severe problems and force the project to start a separate integration sub-project or even worse failure of the entire project. [28]

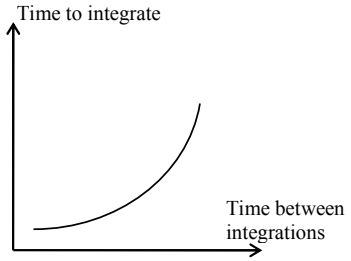

Figure 2: non-linear relationship between the time to integrate and the time between integrations.

Above reasoning and our own experience assert that this relationship is not even a linear [Figure 2], which should encourage frequent integrations even further.

Continuous integration is realized with the help of a tool that automates the build, baselining, testing and reporting. The continuous integration tool is triggered automatically as soon as someone checks in or integrates a code change and provides an automatically updated health status of the code at all times including: code quality, build status and automated tests. A popular continuous integration open source tools is Cruise Control [14].

Automatic Testing - Tools that are triggered when there is something to test, and automatically performs tests on various levels:

- Regression tests
- Unit tests
- Business acceptance tests
- And even GUI tests

Examples of widely used tools to facilitate automatic testing include xUnit frameworks [e.g. 24] and FIT [25]. In XP automatic testing is taken one step further, to test driven development (TDD) [19].

Collective Code Ownership – Traditional strong code ownership divides the code base into modules and assigns each module to a developer, who is responsible for making changes in that module, as well as to uphold the architectural structure of that module. Experience has shown that this often leads to emotional ownership and difficulty in performing changes in other modules, which can lead to delays. Collective code ownership abandons individual ownership of the modules and invites anyone in the team to make changes anywhere [16].

Refactoring - Frequent changes and collective code ownership makes it necessary to repeatedly refactor the code, to keep the structure and architecture simple and avoid the degeneration that occurs over time [26]. Refactoring by anyone in the team is encouraged in agile CM [10].

Iterative and Incremental Development – it is not only agile methods that claim to be iterative and incremental. It is however a main prerequisite for an agile method that the process is iterative, i.e. split up into many smaller parts/deliveries, that adds up incrementally for each iteration. [3]

Retrospective – The iteration retrospective is an agile practice to review the recent iteration to improve routines in the future, covering the whole project scope. With a slightly different scope and not in such a formal manner, the retrospective could advantageously replace the CM audit. [8]

Adaptive Planning – The planning process in agile methods is less formal and tries to be less predictive. Instead an adaptive way of thinking is encouraged. Adaptive planning can also be said to be feedback-driven instead of forecast-driven. The effect this has on CM is that less plans and other formal documentation is created, because focus is on producing code and being adaptive and receptive to change. [29]

Frequent Deliveries – The agile practice to deliver frequently is based on having short feedback loops to the customer and thus being able to respond quickly to changes. Having a lean CM process with little overhead facilitates having short iterations and frequent deliveries.

4. LEAN CM

4.1 CM Performance

The performance of CM in a project or an organization can be measured with two parameters: the degree of CM process formality and the degree of CM (tool) automation. Plotting the two parameters in a diagram gives the following picture [Figure 3]. Here we classify each parameter into relatively high and relatively low measures according to our perception of current practices.

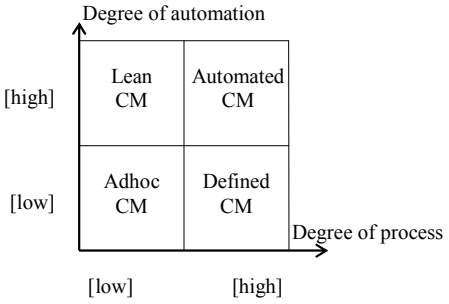

Figure 3: CM performance quadrant.

Ad hoc CM =rel. low degree of defined process and rel. low degree of automation

Defined CM = rel. high degree of defined process and rel. low degree of automation

Automated CM = rel. high degree of defined process and rel. high degree of automation

Lean CM = rel. low degree of defined process and rel. high degree of automation

4.2 Case studies

Many of the ideas to this paper derive from experiences in a number of projects at a few different organizations. We proceed to describe two organizations that have had the greatest impact.

Company A – This is a multinational retail company where an effective global supply chain is a key success factor. IT is a supporting discipline of great importance for fulfilling their supply chain. Company A has adopted a customized RUP adaptation as the central development method for the last few years. Most projects within the organization have interpreted this in a way that most closely resembles a waterfall approach. This makes it a very heavy and defined process with a heavy CM overhead and no tool automation.

Most projects are running the RUP adaptation and have the prescribed dedicated CM role within the project. The same CM resource is often shared between several projects.

The company also uses a gate model to manage their projects and productivity is in general perceived as quite low.

However, also mentioned in coming sections, there are isolated agile initiatives outside of the general enterprise process. This has been indicated as spots in the various quadrants outside the defined CM quadrant associated with Company A in **Error! Reference source not found.**.

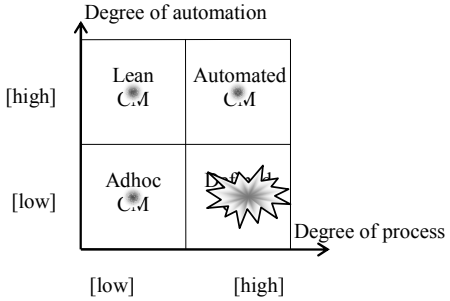

Figure 4: CM performance of company A

Company B – This is a multinational company producing consumer electronic products. IT is a central part of their products, which are very hi-tech, containing advanced custom hardware and software, with short life-cycles. They have a development method that closely resembles waterfall method and a gate model for managing products. They are forced to be productive because of the extremely tough competition in their market. Given the circumstances they succeed quite well, but this is often perceived to be on the staff's expense. However there is plenty of room for improvements, as for example the lead times for change requests are very long.

The CM role in company B is split up into two roles. One which mostly resembles a *change control manager* on a product level, who is occupied with change requests, CCB-issues and release management. The other part of the CM role is heavily biased towards integration.

This company is also firmly classified in the defined CM quadrant as shown in Figure 5.

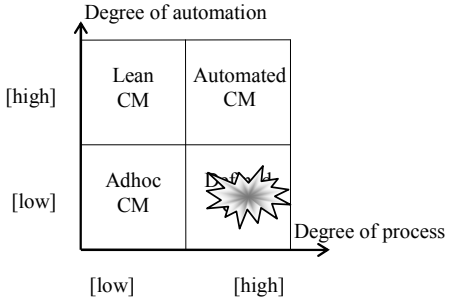

Figure 5: CM performance of company B

4.3 Ad Hoc CM

Ad hoc CM is here defined as when the degree of process is low and the degree of tool automation is low. The typical setup would be having a simple versioning tool, but no dedicated CM role and no formal CM routines. This is sufficient for the smallest of projects and predictable environments.

The degree of process, however, could also be more highly developed, but still be slim and low in waste. This is better, but the lack of automation creates unnecessary manual overhead.

Ad hoc CM in its most extreme form implies no process and no tools, which could also be referred to as "chaos CM" or "no CM". This might be enough for projects with very small teams, maybe only one team member. But in this case there is no traceability, no version handling tool, no support for change management and should generally be avoided because of the potential risk exposure and unexpected high costs these can lead to.

The CM process has for many organizations developed from an initial low degree of process together with a low degree of automation. From there the CM process evolves to a defined CM process concurrently with the evolution of the rest of the software development method, maybe triggered by an initiative to introduce RUP, waterfall, CMM or any other traditional method.

The relation with company A began several years ago, and started with projects categorized as having an Ad Hoc CM process. Between five to ten years ago this seemed to be the standard setup at company A for small and medium sized projects. At the time company A had no central development process and no central function promoting development methods. It was instead up to the individual projects to decide, often ending up with no particularly formal process. This led to a diverse environment and maintenance problems. However the productivity was interestingly enough very high. There are still isolated projects that for some reason have little or no CM process, but in the ones witnessed this have only led to bad traceability and general confusion.

4.4 Defined CM

The defined CM process is defined as when there is a relatively high degree of defined process, but a low degree of tool automation. This could typically be the case when an organization has introduced a quality framework such as CMM or a development method such as RUP. In these methods we have a dedicated CM role with a heavier process including CCB routines.

Company A has the last 5 years been centralizing many of their software systems and functions. As a step in this process they also standardized the development method throughout the organization by introducing RUP as the common development method. The CM role is pretty much taken from off the shelf RUP. This has shifted focus from producing code towards distinct handovers, rigid planning and very formal change management. Unfortunately this has led to a serious decrease in productivity within the organization.

Company B, with its very large development organization, is also considered to be put into this category. The code is very modularized and they have a culture with strong code ownership. This, together with the large number of differently configured products in varying states of development makes CM a

challenging task to handle. The development organization is also large enough for them to require many CMs/integrators in the project in some form of hierarchy that mirrors the architecture of the product. The problem they experience with this setup is that they have very long lead times for change requests.

4.5 Automated CM

We define Automated CM as the rather uncommon situation of having a relatively high degree of CM process while having a relatively high level of tool automation. Our experience says that organizations that benefit from automated tool support have also learned to benefit from agile methods.

At company A, however, there are isolated projects that are moving towards more CM tool automation. This is due to agile supporters among the developers that are introducing continuous integration into a RUP project. This might shift attitudes within the project towards an interest in agile development, but does not do enough for productivity, since the development process is effectively blocking a large part of the possible benefits gained from the automation.

4.6 Lean CM

Lean CM shares the same values as agile development methods and lean principles. Lean CM supports these methods and principles to create increased revenues in a similar way by focusing them explicitly on the CM and the integration of CM towards the rest of the development process, be it agile or not.

Lean CM is achievable when the organization has realized that software development is an empirical process and starts to become more adaptive towards changes while becoming to be less predictive in their planning. To support the agile development process tool use must be automated.

These prerequisites to Lean CM disqualifies traditional development methods like the waterfall or RUP, as these describe a very much more controlled and heavy CM process based on predictive planning with high amounts of waste.

In company A there are a few “underground” initiatives trying to run Scrum. These are framed inside RUP so as not to be disclosed and thereby upsetting management. The product owners in these projects are very pleased with the method and have in general experienced an increase in productivity. In this case however we believe that productivity could increase even further if they could tone down the RUP CM role and introduce continuous integration and perhaps automatic testing.

In company A there is also at least one project that has been completely excepted from the RUP policy. This project is doing XP and is able to deploy into production every 3 weeks. This project is perceived as being very productive.

4.7 Creating Value with Lean CM

It should be clear from the previous sections that even if we run an agile project focusing on being adaptive to change, we still need the benefits from traditional CM to keep control of change in the code. However we need to apply them differently than in traditional projects. We need to help the project create value more rapidly by using lean CM to still remain in control. Therefore, it is time to start talking in terms of return on investment (ROI). To increase ROI we must increase productivity and there are three basic ways to increase productivity [9]:

1. Reduce software development effort.
2. Streamline development processes.
3. Increase customer value.

Let us look at how Lean CM successfully can support the project creating value by analyzing these three basic ways in more detail.

Reducing software development effort

This is basically all about reducing development costs by eliminating investments in features that are not valuable to the customer. To avoid this situation agile methods encourage an empirical approach instead of a waterfall approach forcing detailed and complete requirements upfront. Agile CM can support this by:

Allowing *frequent deliveries* to support short feedback loops to customers, so they don’t have to keep inventing everything up-front and can learn about their needs from actual products rather than paper constructions.

Using a version handling tool is pretty obvious and will not be discussed further.

Streamline development processes

More mature industries have been increasing productivity for decades by streamlining processes and making more efficient use of resources, a good example is the car industry [Toyota]. A mature software development organization is one that can rapidly and repeatedly transform customer requirements into deployed high quality code. This is where lean CM really comes into play by supporting the agile methods and lean principles in order to reduce waste, as well as using tools to further support the process and automate tasks:

Remove the dedicated CM role and let the operative CM tasks be handled by a self-organizing team. To transfer the necessary CM knowledge to the team, the practice of pairing from XP could be used,. i.e. letting the CM, or any other CM experienced person, be a part of the development team and thereby handling the knowledge/experience transfer by pairing with all the other team members in turn, or as appropriate.

Cancel the formal CCB routine and introduce the backlogs of scrum, which has a similar idea but is a more agile way of handling the changing requirements. Badly setup CCB routines just add waste and long lead times to the change process, because important change decisions are held up just waiting for the CCB to convene, This introduces wasted lead time if the decision affects the critical path of the project.

Cancel CM audits and all other excessive QA waste products in order to rely on process and product feedback from retrospectives and customer demos at the end of every iteration.

Continuous integration to get quick feedback loops to the development team about the status of their code.

Automatic testing together with the continuous integration provides further feedback about the status of the code from automated regression tests this is also done by unit tests and integration tests, etc.. Automatic tests provide instant feedback, which is very valuable.

Collective code ownership, where any developer is allowed to change or refactor any code.

Increase customer value

The key issue here is to understand what represents value to the customer, and often the customers do not know this themselves, especially at the start of the project. We must try to understand how the customer will be using the product to be able to deliver value. Since CM is a supporting discipline it doesn't have a great impact on customer value. However the following are still noteworthy practices:

Close collaboration with customers meaning that the customer must be represented on site within the project and the team.

Frequent deliveries and using iterative and incremental development will give the customer quick feedback of the value of the software.

5. METHODOLOGY

This study is an attempt to gather, structure and document industrial Software Engineering experiences gained by software professionals.

The main strategy is to use the skills of a researcher to help professionals to do this in an as a correct manner as is possible without planning the study beforehand. This approach eliminates certain threats to validity compared to other qualitative studies, but of course also introduces others conventional studies do not face. As the professional is very much a part of the whole process, instead of just being a subject at some stage of the research process, errors due to misinterpretations or method are instantly rectified. On the other hand, as the professional is involved in the analysis and writing, completely removing subjectivity is difficult.

The study involves an extremely long period of participatory observations at two case companies denoted Company A and B, combined with an overview of current state of CM as documented in the literature and in the agile community.

Countermeasures to increase validity of the conclusions made have been applied as far as is possible as shown in Section 5.1.

5.1 Threats and Countermeasures

We present and try to eliminate threats to validity of the experiences and subsequent conclusions presented in the paper according to Lincoln and Guba model [23] summarized in Table 1.

Table 1: Strategies for dealing with threats to validity [23].

	Threat to validity		
Strategy	**Reactivity**	**Researcher bias**	**Respondent bias**
Prolonged involvement	Reduces threat	Increases threat	Reduces threat
Triangulation	Reduces threat	Reduces threat	Reduces threat
Peer debriefing	No effect	Reduces threat	No effect
Member checking	Reduces threat	Reduces threat	Reduces threat
Negative case analysis	No effect	Reduces threat	No effect
Audit trail	No effect	Reduces threat	No effect

The professional Software Engineer has been involved long-term with the case study companies, for over ten years in total, while the researcher is only familiar to the case companies through discussions with various people. This is assumed to reduce researcher bias threats while maximizing involvement. As the experience gathering has been done without prior planning the study has a threat du to a lack of structure in the experience gathering. This is countered by the use of *triangulation* and *peer debriefing*.

Triangulation is attained through data triangulation by comparing multiple data sources: own experiences, observations and collected experience in publications and the agile community.

Peer debriefing is not used to any larger extent in the study, Member checking on the other hand is used, by feeding back analysis results to other people working at the case companies and incorporating their comments in the conclusions.

No *audit trail* strategy is used in the study.

The large number of projects currently using agile methods and CM implies that *transferability* should be high, although case studies are always coloured by their specific context.

As a continuation of the work presented here, it would be reasonable as to verify experiences of cases in other quadrants of the Lean CM structure presented.

In summary, reasonable countermeasures to validity threats in an experience paper have been implemented.

6. CONCLUSION

The recipe for a lean CM process for agile projects is to make use of the benefits of traditional CM, without having the drawbacks of a heavy controlling process. The principal properties identified for Lean CM are:

- Cancel the formal CCB routine and handle changes in a Scrum backlog.
- Cancel the CM audit and handle the audit in the iteration retrospective.
- Automate as much as possible by introducing continuous integration and automated testing.
- Remove the dedicated CM and let the team handle the daily CM tasks
- Educate the agile team in necessary CM skills by temporarily pairing the team members with a person skilled in both CM and agile methods.

The experiences presented in this paper suggest that projects inspired to introduce Lean CM can experience increased ROI as several ROI indicators are affected, even if the project does not simultaneously introduce agile methods as a whole.

7. ACKNOWLEDGMENTS

Anders Sixtensson and Per Runeson for providing feedback and inspiration.

8. REFERENCES

[1] Kent Beck, *eXtreme Programming - Embracing Change*, Addison Wesley, 1999.

[2] Mike Beedle and Ken Schwaber, *Agile Software Development with SCRUM*, Prentice Hall, 2001.

[3] Craig Larman, *Agile & Iterative Development, A Manager's Guide*, Addison Wesley, 2004.

[4] Mary Poppendieck and Tom Poppendieck, *Lean Software Development: An Agile Toolkit for Software Development Managers*, Addison Wesley, 2003.

[5] Alistair Cockburn, *Agile Software Development*, Addison Wesley, 2002.

[6] Ivar Jacobsson, Grady Booch and James Rambaugh, *The Unified Software Development Process*, Addison Wesley, 1999.

[7] David E. Bellagio and Tom J. Milligan, *Software Configuration Management Strategies and IBM Rational ClearCase*, IBM Press, 2005.

[8] Norman L. Kerth, *Project Retrospectives: A Handbook for Team Reviews,* Dorset House Publishing Company, 2001.

[9] Poppendieck LLC, "Software Development Productivity", http://www.poppendieck.com/publications.htm

[10] Martin Fowler, Kent Beck, John Brant, William Opdyke and Don Roberts, *Refactoring: Improving the design of exisiting code*, Addison Wesley, 1999.

[11] IEEE-Standard-729-1983, *Standard Glossary for Software Engineering, ANSI/IEEE.*

[12] The SEI Software Capability Maturity Model

[13] Kent Beck et al, *Manifesto for Agile Software Development*, http://www.agilemanifesto.org, 2001.

[14] *CruiseControl*, http://cruisecontrol.sourceforge.net/

[15] Peter H. Feiler, "Configuration Management Models in Commercial Environments", CMU/SEI-91-TR-7, Carnegie-Mellon University/Software Engineering Institute, March 1991.

[16] Martin Fowler, *Code Ownership*, http://www.martinfowler.com/bliki/CodeOwnership.html, 2006.

[17] Philippe Kruchten, *The Rational Unified Process: An Introduction*, Addison Wesley, 2003.

[18] Carnegie Mellon Univ. Software Engineering Inst. *The Capability Maturity Model: Guidelines for Improving the Software Process*, Addison Wesley, 1995.

[19] Kent Beck, *Test Driven Development: By Example*, Addison Wesley, 2002.

[20] Stapleton, J. *Dynamic Systems Development Method: The Method in Practice*, Addison Wesley, 1997.

[21] Mary Poppendieck, *Agile Contracts: How to Develop Contracts that Support Agile Software Development*, Agile 2005 Conference, 2005.

[22] Laurie Williams, Robert R. Kessler, Ward Cunningham and Ron Jeffries, *Strengthening the Case for Pair-Programming*, IEEE Software (Vol. 17, No. 4), 2000.

[23] Robson, C., *Real World Research*, Blackwell Publishers, Oxford, 2002.

[24] JUnit testing framework http://junit.org, 2006.

[25] Framework for Integrated Test http://fit.c2.com, 2006.

[26] Lorin Hochstein, Mikael Lindvall, *Diagnosing architectural degeneration, sew*, p. 137, 28th Annual NASA Goddard Software Engineering Workshop (SEW'03), 2003.

[27] Goldratt, E., *Critical Chain*, North River Press, 1997.

[28] Craig Larman, *Agile & Iterative Development, A Manager's Guide*, p. 60, Addison Wesley, 2004.

[29] Martin Fowler, *The New Methodology,* http://www.martinfowler.com/articles/newMethodology.html , 2005