The Lean Gap: A Review of Lean Approaches to Large-Scale Software Systems Development

J.Pernstål[1], T. Gorschek[3], R. Feldt[2]

[1]*Volvo Car Corporation, SE-405 31 Göteborg, Sweden*
jpernsta@volvocars.com

[2]*Department of Computer Science and Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden*
robert.feldt@chalmers.se

[3]*Blekinge Institute. of Technology, Karlskrona, SE-372 25*
tony.gorschek@bth.se

Abstract

Lean approaches to product development (LPD) have had a strong influence on many industries and in recent years there have been many proponents for lean in software development as it can support the increasing industry need of scaling agile software development. With it's roots in industrial manufacturing and, later, industrial product development, it would seem natural that LPD would adapt well to large-scale development projects of increasingly software-intensive products, such as in the automotive industry. However, it is not clear what kind of experience and results have been reported on the actual use of lean principles and practices in software development for such large-scale industrial contexts. This was the motivation for this study as the context was an ongoing industry process improvement project at Volvo Car Corporation and Volvo Truck Corporation.

The objectives of this study are to identify and classify state of the art in large-scale software development influenced by LPD approaches and use this established knowledge to support industrial partners in decisions on a software process improvement (SPI) project, and to reveal research gaps and proposed extensions to LPD in relation to its well-known principles and practices.

For locating relevant state of the art we conducted a systematic mapping study, and the industrial applicability and relevance of results and said extensions to LPD were further analyzed in the context of an actual, industrial case.

A total of 10,230 papers were found in database searches, of which 38 papers were found relevant. Of these, only 42 percent clearly addressed large-scale development. Furthermore, a majority of papers (76 percent) were non-empirical and many lacked information about study design, context and/or limitations. Most of the identified results focused on eliminating waste and creating flow in the software development process, but there was a lack of results for other LPD principles and practices.

Overall, it can be concluded that research in the much hyped field of lean software development is in its nascent state when it comes to large scale development. There is very little support available for practitioners who want to apply lean approaches for improving large-scale software development, especially when it comes to inter-departmental interactions during development. This paper explicitly maps the area, qualifies available research, and identifies gaps, as well as suggests extensions to lean principles relevant for large scale development of software intensive systems.

Keywords

Systematic Mapping Study; Software Engineering; Lean Product Development; Lean Software Development; Agile Software Development; Automotive Software Development; Software Intensive Product Development; Process Improvement.

1 Introduction

Software is rapidly becoming a substantial component and seen as the main driver and source of innovations in a number of traditionally hardware-focused industries, (e.g., automotive and aerospace) (Vekantesh-Prasad et al. 2010; Broy et al. 2007). For example, the worldwide value of automotive software-intensive systems is expected to rise from 127 billion Euros in 2002 to 316 billion Euros in 2015 (Dannenberg and Kleinhans 2004). In these organizations, but generally in large organizations, the software-intensive systems are commonly developed in the context of large-scale development (i.e. systems of systems development), where software constitutes only one, but important, part of the whole (Nihtilä 1999; Broy et al. 2007). This context is of particular interest in this study where one of the key challenges is to integrate the software development into the overall and multidisciplinary development of the complete product (Nihtilä 1999). This is a challenge at Volvo Car Corporation (VCC) and Volvo Truck Corporation (VTC), where the advancement of software has increased the uncertainty due to its changeable nature and the interdependences between development tasks and artifacts, elevating the complexity of their organizational structure and leading to communication and coordination problems across departments (inter-departmental).

To avoid, or counter, large overhead, and enabling being highly responsive to change, software development organizations have turned to agile software development over the past years, yielding some good results (e.g., Mannaro et al. 2004; Layman et al. 2004; Svensson and Höst 2005), but also leaving questions unanswered (e.g., Abrahamsson et al. 2003; Boehm 2002; Conboy 2009; Wellington et al. 2005), ranging from scalability in large organizations to actual productivity and quality issues. Thus simply applying agile (e.g., Scrum (Schwaber and Beedle 2001)) as a stand-alone solution to coordination and communication problems in such large and complex organization as VCC and VTC would not work. Hibbs et al. (2009) and Petersen (2010) claim that lean practices and principles, building on lean product development (LPD), in contrast to agile, can be applied to any scope, and is a prerequisite for scaling agile due to the unique focus on the whole.

For all its benefits, LPD stems from the Toyota Product Development System (TPDS) (Morgan and Liker 2006) for managing hardware development, and that was not originally designed for development of software-intensive systems. Rather, it has primarily evolved, and been applied to development of products that have traditionally been highly modular (e.g., vehicles) enabling independent development and manufacturing (Morgan and Liker 2006). However, these products are now becoming more large and complex, integrated systems due to the increased amount of interacting software-intensive systems (e.g., central locking and engine control in a car) (Broy et al. 2007). Even though Poppendieck and Poppendieck (2003) have looked into how lean principles and practices can be used in software engineering (SE) and presented lean software development (LSD), it is unclear what empirical evidence exists that lean principles and practices can be successfully applied for software development in large-scale development projects of software-intensive products. It is therefore relevant to review previous work and collect evidence of the feasibility of applying lean to such industrial contexts from both an industrial and an academic perspective.

The study presented in this paper was primarily motivated by an industry need of locating and evaluating state of the art that could be contributing for solving a number of key improvement issues identified in a previous case study (Pernstal et al. 2012). The case study is part of an SPI project, assessing the inter-departmental interaction between Product development (PD) and Manufacturing (Man) in the context of large-scale software-intensive systems development at two Swedish automotive companies, namely Volvo Car Corporation (VCC) and Volvo Truck Corporation (VTC). Since both companies have implemented lean manufacturing and are adopting LPD, they expressed a need of identifying the state of the art in large-scale software development building on lean principles and practices. Consequently, the main objective of the study presented in this paper is to evaluate and summarize such state of the art for ensuring that this knowledge is not omitted when developing solution for the identified issues in the companies. In addition we also aim to identify needs and opportunities for future research—in essence figuring out what new challenges and inadequacies might be present given the context of the case. For this we conducted an extensive Systematic Mapping Study (SMS) (Petersen et al. 2008) as our initial searches in database showed that there were relative few relevant and high-quality studies on the topic of interest. Primarily because the case companies are adopting the well-established LPD principles originating from TPDS and described in Morgan and Liker (2006), we structure the main part of the analysis of the results according to these principles in order to identify lean gaps in large-scale software-intensive development. Furthermore, we could have used the principles for LSD, but we wanted to take a broader view, as such development covers different departments and engineering disciplines and is not limited to software development. The LPD principles constitute the core of, and serve as comprehensive guidelines for companies in their efforts to achieve LPD where one of the key challenges is to obtain LPD across the whole company—not only within the development organization, but also in other surrounding organizations such as marketing, product planning, purchasing and manufacturing. In addition, industries developing software-intensive systems have implemented lean manufacturing principles originating from the Toyota Production System (TPS) (Liker 2004; Ohno 1988). In order to minimize waste in their lean manufacturing processes, a well-known trend among these industries is also to improve their development processes by implementing related LPD principles (Morgan and Liker 2006)—but it is unclear how the principles address the fact that software as an artifact, and software engineering as a discipline, are becoming a central component in the products developed.

The paper maps the area, identifies relevant work and qualifies it in terms of quality, and reveals knowledge gaps. This is then discussed and reflected upon based upon previous observation from needs identified at the case companies VCC and VTC, with the purpose of helping the case companies in their decisions of adopting lean principles and practices as well as showing the challenges in doing so. In addition, based on the findings and industrial needs we propose extensions to the lean principles in the running analysis.

The remainder of the paper is organized as follows. Section 2 gives a brief background and summarizes the related work. Section 3 provides an outline of the research methodology used in the SMS. The results of the study are presented together with analysis in Section 4. Finally conclusions and future research directions are presented in Section 5.

2 Background and Related Work

This section describes large-scale software development, presents related work on LPD and summarizes previous most relevant reviews of studies reporting experiences, and best practices within, or close to, the scope of this SMS. Finally, motivations and objectives are given.

2.1 Large-Scale Software-Intensive Systems development

Challenges in software development are often related to scaling-up software-intensive systems as the complexity increases more than linear with the size (Brooks 1988; Curtis et al. 1988; Kraut and Streeter 1995). Many industries develop large software systems that are embedded in their products (e.g., vehicles and avionics) where the complexity becomes even more manifested (Vekantesh-Prasad et al. 2010; Broy et al. 2007). This because such products are typically built of functions, systems, and sub-systems including various hardware components and running on a large amount of interacting software, requiring precise coordination and integration of development tasks across multiple departments and engineering disciplines (Broy et al. 2007). For example, the logics of the central locking system (CLS) is controlled by software distributed over different sub-systems of the complete vehicle (e.g., door modules and alarm system), involving mechanical engineering (e.g., door locks), electrical engineering (e.g., lock motors. sensors and cable harnesses) and SE (e.g., controlling of the locking function logics). In addition, manufacturing engineering must be involved in order to secure that the system fit the manufacturing processes—how shall, for example, the CLS be configured and quality assured in manufacturing? Moreover, in these industries, many of the systems are safety critical (e.g., airbag and anti-locking braking systems in a vehicle) where the development is governed by safety standards (e.g., ISO 26262), incorporating such as the V-model (ABG 1997) and including, for example, hazard analysis, safety analysis and verification). For managing the rapidly increasing share of software in these products, it has been acknowledged in research that the role of SE needs to be better integrated into the PD as a whole (Nihtilä 1999).

2.2 Lean Product Development

Based on several studies on the automobile industry by researchers at the International Motor Vehicle Program (IMVP) at the Massachusetts Institute of Technology (MIT), Womack et al. (1990) introduced the LPD concept. This concept is building on TPDS under the heading of the broader concept of lean production. They concluded that there were differences between mass and Japanese lean producers not only in the manufacturing processes, but also in the PD processes in terms of practices regarding strong leadership, teamwork, early communication and coordination across departments, and concurrent development.

Many companies developing large software-intensive systems have implemented lean manufacturing, but to make full use of this competitive weapon—squeezing out more waste of lean manufacturing processes—lean needs also to be extended to the PD processes (Morgan and Liker 2006). To increase the effectiveness in the PD processes, several companies have started to implement some inherent principles and practices of LPD. Continuous improvement (Kaizen), Kanban, concurrent engineering, customers and suppliers' involvement, visual management, group work and cross-functional teams emerge as some of the practices used to reach the purpose of LPD (Karlsson and Åhlström 1996; Morgan and Liker 2006;Sobek et al, 1999; Wang et al. 2012).

However, deploying only a few of the practices is not enough for achieving LPD. Womack and Jones (2003) claim that lean is a way of thinking that must be adopted throughout the whole enterprise. They conceptualize lean thinking into five categories: value, value stream, flow, pull, and perfection. Value defines the use that a product offers a customer, and works backward to build business processes. A value stream describes each step in processes and categorizes them with regard to the value added (e.g., value adding, necessary non-value adding and non-value adding steps). Flow organizes processes so products move smoothly through the value-creating steps. Pull involves each customer calling output from the previous step, on demand. Finally, perfection entails continuous improvement of processes for meeting customer needs and with zero defects.

Literature specifically addressing LPD, are typically referring to studies on TPDS (Karlsson and Åhlström 1996; Morgan and Liker 2006; Kennedy et al. 2008). Morgan and Liker (2006) present a comprehensive work on LPD. Using the Sociotechnical Systems Theory (STS) (e.g., Miller and Rice 1967) and the principles and practices of TPDS, they describe the core and essence of LPD in a model called Lean Product Development System (LPDS). LPDS is based on the idea that LPD is a philosophy being adopted throughout the whole enterprise rather than superficial applications of a few lean principles and practices to parts of an organization. The LPDS model contains three primary sub-systems: (1) process, (2) skilled people and (3) tools and technology. These are described by means of 13 principles, see Fig. 1.

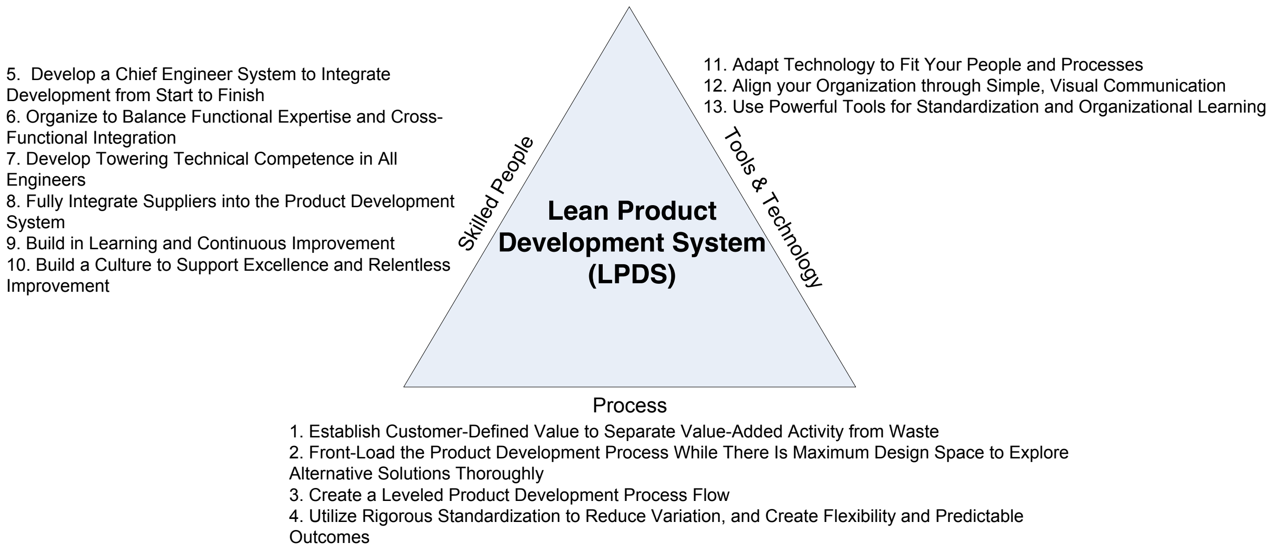
5. Develop a Chief Engineer System to Integrate Development from Start to Finish
6. Organize to Balance Functional Expertise and Cross-Functional Integration
7. Develop Towering Technical Competence in All Engineers
8. Fully Integrate Suppliers into the Product Development System
9. Build in Learning and Continuous Improvement
10. Build a Culture to Support Excellence and Relentless Improvement
Skilled People
Lean Product Development System (LPDS)
Tools & Technology
11. Adapt Technology to Fit Your People and Processes
12. Align your Organization through Simple, Visual Communication
13. Use Powerful Tools for Standardization and Organizational Learning
Process
1. Establish Customer-Defined Value to Separate Value-Added Activity from Waste
2. Front-Load the Product Development Process While There Is Maximum Design Space to Explore Alternative Solutions Thoroughly
3. Create a Leveled Product Development Process Flow
4. Utilize Rigorous Standardization to Reduce Variation, and Create Flexibility and Predictable Outcomes

Fig. 1. LPDS model adapted from Morgan and Liker (2006).

Whereas LPD principles are viewed as a platform for implementing lean approaches to PD on an enterprise-level, LSD is commonly seen as a method for applying lean in the software development discipline and referred to the work presented by Poppendieck and Poppendieck (2003). Their adaptation of lean principles into seven software development principles is the main source for interpreting lean principles in the context of SE. These principles are:(1) eliminate waste—doing only what adds customer value without delays, (2) amplify learning—using frequent feedback loops, (3) delay commitment—deciding as late as possible, (4) deliver as fast as possible—minimizing the time from receiving customers' needs to delivery, (5) empower the team—fostering respect for people among leaders and staff and building expert technical workforce, (6) build integrity in—establishing product quality as early as possible for avoiding defects in late phases, and (7) see the whole—for avoiding sub-optimizations, the whole software development process is considered.

LSD has been associated with agile methods, such as XP (Beck, 2004) and Scrum (Schwaber and Beedle 2001), and is often seen as just another agile method (Dybå and Dingsøyr, 2008; Higsmith, 2002). However, the definitions of the terms agile and lean are often ambiguous and inconsistent in the software development literature (Conboy, 2009). This makes it difficult to identify differences and overlaps between them, but in recent literature LSD is acknowledge as being a method category with its own identity and even claimed to be the next evolutionary step from agile towards lean approaches in software development (Hirnabe, 2008; Wang et al., 2012). Poppendieck and Poppendieck (2003) advocate that lean is a platform upon which to build agile software development practices, and view lean principles as the theoretical foundation behind agile software development. Furthermore, Hibbs et al. (2009) claims that lean principles can be applied to the whole enterprise where software development constitute one part of large-scale PD processes, while agile methods mainly focus on team levels activities and specific practices for developing software and usually do not concern the surrounding business context in which software development take place. Others argue that even though there are differences, but also overlaps, between agile and lean principles, they complement each other, and in particular, the unique focus on the whole in LSD supports the expanding industry need of scaling agile software development (Coplien and Bjornwig, 2010; Petersen, 2010; Wang et al., 2012). Similarly, it is commonly claimed that agile methods have their shortcomings in large-scale development, where it is recommended to mix the best features of traditional plan-driven and agile methods (e.g., Boehm, 2002; Conboy, 2009; Karlström and Runesson, 2005; Sommerville, 2007).

In summary, earlier work on lean and agile in SE indicates that adoption of lean principles in software development is beneficial especially when it comes to large software systems development. Even though lean and agile methods are influencing also more traditional industries, such as the automotive one, the transition to lean and agile methods have only started and is not yet widespread in the context of large-scale software development (Dybå and Dingsøyr, 2008; Wang et al., 2012).

2.3 Summary of Related Reviews

For locating related literature reviews, we searched the scientific databases ACM Digital Library, IEEE Xplore, Inspec, ISI Web of Science, Scopus, and Google Scholar. The search string was based on the synonyms for systematic review defined by Biolchini et al. (2007). Reviews covering the area focused on in this paper could not be found. However, studies reviewing literature on lean and agile methods focusing on the SE field were found. The following presents the most relevant ones.

Dybå and Dingsøyr (2008) conducted a systematic literature review (SLR) of empirical studies of agile software development and LSD up to 2005, and identified 36 relevant empirical studies. Of those, only one reported on applying lean practices to software development. Other main findings were that most of the empirical studies focus on a single development method (e.g., XP) and studies on agile), and implementations are mainly carried out on smaller scale (only three of the papers investigated settings with more than 50 people).

Cawley et al. (2010) performed an SLR investigating to what extent lean and agile software development methods have been adopted in regulated safety critical systems development. Most of the studies identified were based on agile practices (XP and Scrum) combined with traditional plan-driven development methods, but they found no studies where LSD had been used. However, they believe that LSD has a potential of improving the development of safety-critical systems, and thus, point out the need of further investigations in this area.

Wang et al. (2012) reviewed 30 experience reports published in agile conferences in which lean principles and practices had been applied to agile software development. They divided the reports into six categories of lean applications in agile software development. One of those concerned applications of lean approaches for improving the interaction with other units that had already implemented lean principles and practices established in the overall PD process while keeping the agile software development processes internally. Furthermore, they found that several recently published papers reporting on mature agile organizations show that these organizations have a tendency to move from time-boxed agile processes to more flow-based lean processes. The growing interest in LSD is also reflected by the fact that special issue on lean has recently been published in IEEE software (2012).

2.4 Motivations and Objectives of this SMS

There are three main rationales for carrying out the SMS presented in this paper: (1) to the best of our knowledge there are no systematic mapping studies locating the state of the art building on LPD with a focus on large-scale software development, (2) an industrial need of evaluating the

strength of evidence and potential industrial value of such state of the art, and (3) gaps and needs for future research in the area are unclear.
The SMS presented in this paper differs from previous reviews. Dybå and Dingsøyr (2008) included LSD and had a clear focus on agile methods, while Wang et al. (2012) limited their review to state of the art using lean applications in agile software development. Our main focus is on lean application to large-scale software intensive systems development, mainly because agile methods have primarily been applied to and studied in small-scale software development projects (Dybå and Dingsøyr, 2008), which is a problem for large-scale development. Lean approaches on the other hand seem to scale better than agile (Wang et al. 2012), which is why we use lean and lean principles as the base for our study and analysis. We analyze our results from a broad lean perspective by using LPD principles applicable to the whole enterprise and not the LSD principles primarily adapted for software development as the main focus here is on software developed in the context of larger systems in a multidisciplinary setting (e.g., software controlling the engine or the CLS system in a vehicle).
Assessing the methodological quality and the strength of evidence in order to assist practitioners in the case companies to evaluate the potential benefits and risks, and decision support prior to adopting the state of the art is another main reason for performing the SMS. For this we reflect on our results from the viewpoint of inter-departmental interaction by building on the collected data and findings in an industrial case study reported in Pernstal et al. (2012). The study examines the interface between PD and Man in large-scale development of software-intensive systems at VCC and VTC. PD is concerned with design and development of software-intensive automotive systems (e.g., development of power train and chassis control systems for vehicles). Man is concerned with managing these systems when producing vehicles (e.g., vehicle manufacturing operations affected by power train and chassis control systems). The case was chosen mainly because inter-departmental coordination and communication is a key challenge in large-scale software projects (Kraut and Streeter, 1995) and the interaction between PD and Man has been identified as critical, both in the case companies and in literature (Morgan and Liker, 2006; Pernstal et al., 2012; Wheelwright and Clark, 1994; Nihtilä, 1999). Furthermore, the researchers involved in the study presented in this paper have access to rich and detailed information about the case, allowing better possibilities to judge the industrial applicability and relevance of the results. In addition, we assessed the quality of the selected papers by using an evaluation model proposed by Ivarsson and Gorschek (2009, 2011) for gauging rigor and industrial relevance of studies.
Another main reason for conducting the SMS is to identify gaps and opportunities for future research (Kitchenham and Charters 2007). LPD has primarily evolved and been adopted in traditionally hardware-focused industrial sectors (e.g., automotives (Ward, 2007)) and aerospace (Lean Aerospace Initiative (LAI) (Murman, 2004)), while the empirical evidence for the applicability of lean and agile software development methods to such large-scale industrial contexts is unclear (Dybå and Dingsøyr, 2008). However, software is increasingly becoming an important component in these sectors, and unlike much hardware development and manufacturing, software development is a nonroutine activity and the "material" itself is intangible, changeable, and unpredictable (Brooks, 1987; Kraut and Streeter, 1995). Furthermore, to identify and suggest areas for further studies addressing LPD in software-intensive industrial sectors, we give an overall picture and an in depth analysis of how the state of the art in large-scale software development based on lean principles and practices are related to LPD. In addition, to outline research needs from an SE perspective, the identified state of the art is mapped to the different knowledge areas (KA) as defined by SWEBOK (Abran et al., 2004). We chose SWEBOK as a way to structure the

paper as it is well known and established framework sponsored by IEEE. Although not perfect or seen as optimal by all (Kaner, 2003)—we considered it to be adequate for our purposes, and any framework used have both proponents and opposers.

3 Research Methodology

The process for the SMS presented in this paper follows the guidelines by Kitchenham and Charters (2007). It consists of the four main steps: (1) definition of research questions, (2) generation of search strategy, (3) study selection, and (4) data extraction and quality assessment. Three researchers were involved in this SMS. In order to enhance the validity and reliability, we have continuously documented and updated the research procedures in a review protocol, which is summarized below.

3.1 Research Questions

The research questions posed in the SMS presented in this paper are shown in Table 1.

Table 1.
Research questions.

Research question	Description
RQ1: What is the state of the art in large-scale software development primarily based on lean principles and practices?	Investigating if there are research contributions based on lean principles and practices in large scale development of software-intensive systems. In this study, we impose no limitations with regard to the level of evaluation of the state of the art.
RQ2: What are the characteristics of the identified state of the art?	Structuring and analyzing the main contributions of the publications by combining the properties included in the sub-questions RQ2.1, RQ2.2, RQ2.3 and RQ2.4.
RQ2.1: What type of research is commonly conducted?	Reflecting the research approaches used in the publications independent from the specific focus area.
RQ2.2: What is the relevance of the state of the art?	Assessing the quality of the results reported by examining the level of the industrial relevance.
RQ2.3: What is the rigor of the state of the art?	Assessing the quality of the results reported by examining the level of the rigor (e.g. descriptions of study design and context).
RQ2.4: What topics in SE does state-of-the-art explicitly and clearly target?	Capturing the topics covered and identifying research gaps in the field of SE.
RQ3: What relationships are there between state of the art and the principles of LPD?	Ordering and analyzing of the state of the art reported in relation to the LPD principles. The objectives are to identify research gaps and needed extensions in LPD for large-scale software development, and evaluate the applicability of the state of the art, giving the industrial partners support in decisions on the SPI project.

3.2 Search Strategy

The SMS was limited to peer-reviewed conference papers or journal articles written in English language, published between 1990 to 2010, since the term 'lean' was first coined by Womack et al.(1990). Furthermore, in order to reduce the number of irrelevant results, the search was only applied to the title, keywords and abstract (Dybå et al. 2007).

The search string was generated on the basis of the scope of this SMS and is composed of two groups of search terms, population AND intervention. Population includes alternative keywords representing creation of software-intensive systems or products. Keywords covering common concepts, principles and practices related to lean are embedded in intervention primarily based on (Morgan and Liker 2006; Karlsson and Åhlström 1996; Sobek et al. 1999; Wang et al. 2012). As inter-departmental interaction is one of the most critical challenges in large-scale software development (Kraut and Streeter 1995), principles and practices that we could relate to this were specifically considered. In order to identify relevant keywords and balance the comprehensiveness and precision, the search string evolved over several pilot searches in the Scopus database. These searches showed that terms not specific for lean, such as "integrated product development", was used in publications that could be relevant (e.g., Negroni and Trabasso 2009). However, these studies were excluded during selection process, primarily because they did not focus on software development. To ensure the reliability and relevancy of the searches and to evaluate the search strings, emerging key publications were listed and compared to the trial search.

In the first trial search, population targeted large-scale software development. However, the search showed that there was a lack of relevant publications in this specific area (less than five). To progress the review, we decided to extend the scope of the SMS so population covered development of software-intensive systems in general. The relevance of the located state of the art for large-scale software development was then assessed during the data extraction step.

With regard to the quality of the studies, it would have been effective to use keywords that limit the publications to studies where the state of the art has been evaluated in some way (e.g., empirical, experience, lesson learned etc.) or in terms of a specific research method (e.g., experiment, case study etc.). However, trial searches with such restrictions on the search string showed that there was a high risk of missing relevant papers, resulting in an incomplete overview of the reviewed area. As a consequence, this delimitation was not imposed. Moreover, the keywords for some of the practices commonly used in lean, such as pull and problem solving, were too general, generating many irrelevant hits, and were thus discarded. The final search string's keywords used for population and intervention and the Boolean expression are presented in Table 2.

Table 2.
Search string and keywords.

Population	**AND**	**Intervention**
software AND (development OR engineering OR embedded system* OR electronic*)		lean* OR Kanban OR Kaizen OR 'continuous improvement' OR 'cross* functional' OR 'concurrent engineering' OR 'integrated product development'

The search string was applied to five electronic databases in the SE field. The list of selected databases and the dates of the electronic search are provided in Table 3.

Table 3.
Searched databases.

Databases	**Search date**
ACM Digital Library	2011-02-05
IEEE Xplore	2011-02-05
Inspec	2011-02-05
ISI Web of Science	2011-02-05
Scopus (comprises Compendex)	2011-02-05

The search in the databases yielded in a total of 13,984 hits. After removing duplicates 10,230 publications remained to be further investigated. Due to large amount of publications, the reference management application Mendeley (http://www.mendeley.com) was used.

3.3 Study Selection

The selection of relevant studies that are within the scope and relate to the research questions posed in this SMS was based on a number of criteria for including publications. The purpose of the criteria is to ensure inclusion of studies on large-scale software development and any kind of lean development. In Table 4, the selection question and inclusion criteria are specified and briefly explained.

Table 4.
Criteria for study selection.

Selection question	Inclusion criteria	Explanation
Is the main focus of the study relevant?	The paper clearly states that it focuses on **development** of **software or software intensive products or systems** AND includes any kind of **lean development**	-Software-intensive systems or products consist of integrated software and hardware solutions, e.g., automobiles, aircrafts, mobile phones, etc. - Lean development involves methods such as, Kanban, Kaizen (continuous improvement), concurrent development etc.

The process for study selection began with selecting relevant publications based on screening the titles, keywords, and abstracts of the papers.

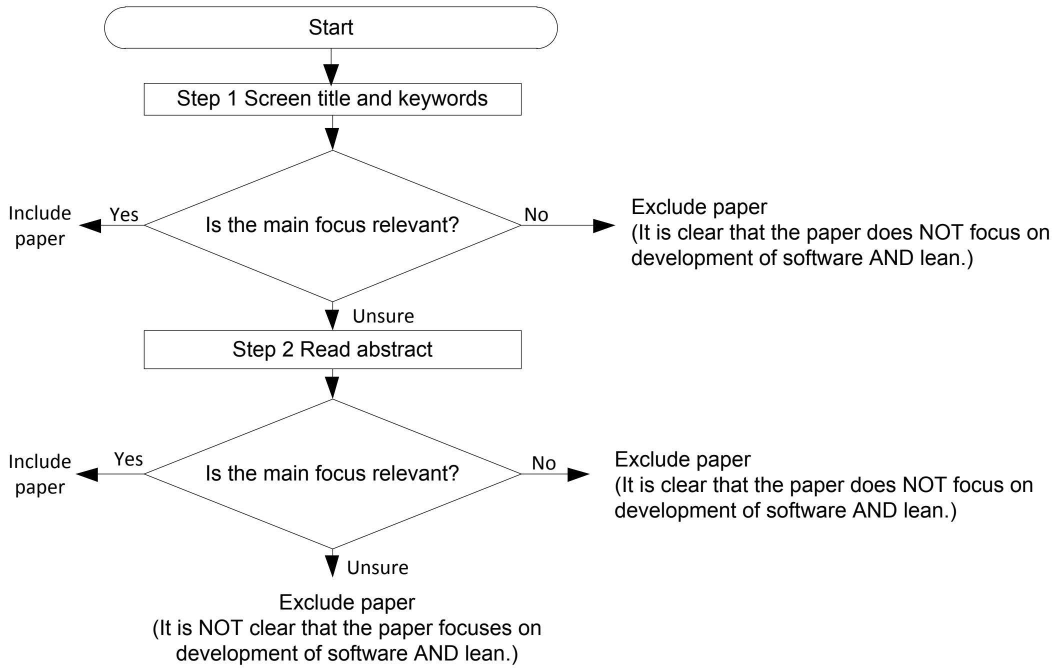
Start
Step 1 Screen title and keywords
Is the main focus relevant?
Yes
Include paper
No
Exclude paper (It is clear that the paper does NOT focus on development of software AND lean.)
Unsure
Step 2 Read abstract
Is the main focus relevant?
Yes
Include paper
No
Exclude paper (It is clear that the paper does NOT focus on development of software AND lean.)
Unsure
Exclude paper (It is NOT clear that the paper focuses on development of software AND lean.)

Fig. 2. Process for study selection based on screening the titles, keywords, and abstracts.

As shown in Fig. 2, the screening process consists of two main steps. First, the title and keywords are screened. Depending on whether the main focus of the study is relevant, the paper is included. If there are uncertainties about the main focus of the study, the abstract is read in the second step where the paper is only included if the main focus is clearly relevant.
The three researchers elaborated a manual containing the process and selection criteria. To enhance the quality of the manual and establish a unified understanding and interpretation of the criteria and the process, we piloted and computed the inter-rater agreement. In the first pilot, we assessed 100 randomly selected publications from the search performed in the databases. The

Fleiss' Kappa (Fleiss 1971) value showed a moderate agreement (0.5) according to Landis and Koch (1977). We scrutinized the results of the pilot and refined the manual. In particular, the definitions of the selection criteria were perceived too implicit, and hence they were further detailed, specified and clarified by adding explanations. Then we performed a second pilot on 30 randomly selected publications, yielding a substantial agreement (0.74) (Landis and Koch 1977), and one of the researchers used the agreed manual on the remaining papers selected for the screening.
After the screening, one of the researchers also applied the inclusion criteria to the conclusion and introduction sections of the remaining set of papers. Papers that were difficult to judge (e.g., unclear focus on lean approaches to software development) were classified as "unsure". A full-text reading of these papers was conducted by the three researchers, and then further evaluated and discussed based on the inclusion criteria (see Table 4) in a consensus meeting, yielding a final set of agreed papers for data extraction. Fig. 3 shows the overall study selection process and the statistics for how the publications from the selection based on the search in databases were reduced to a final set of studies accepted for data extraction and quality assessment.

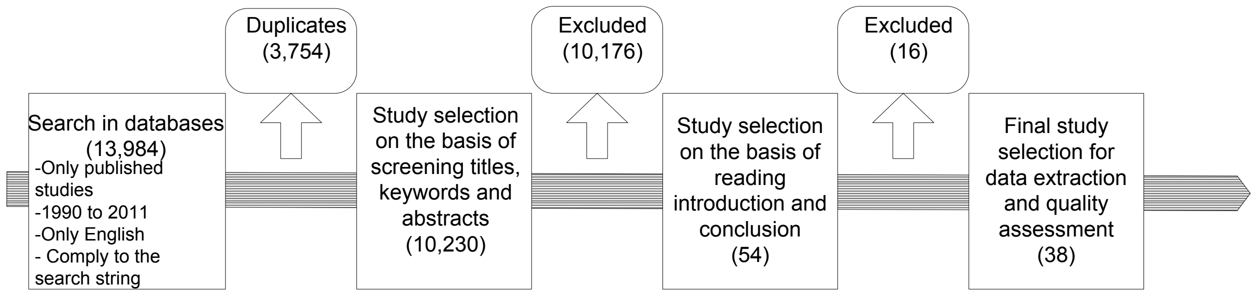
Duplicates (3,754)
Excluded (10,176)
Excluded (16)
Search in databases (13,984)
-Only published studies
-1990 to 2011
-Only English
- Comply to the search string
Study selection on the basis of screening titles, keywords and abstracts (10,230)
Study selection on the basis of reading introduction and conclusion (54)
Final study selection for data extraction and quality assessment (38)

Fig. 3. Overall study selection process and statistics.

The search string generated 13,984 published studies between 1990 and 2011 written in English. Of those, 3,754 were identified as duplicates by listing titles and authors alphabetically in Mendeley, which resulted in leaving 10,230 publications for the screening. After applying the inclusion criteria on the titles, keywords, and abstracts, 10,176 papers were found irrelevant, reducing the remaining number of publications to 54. Many of the excluded papers reported on engineering of computer-aided software applications and tools applied to virtual development of mechanical parts (e.g., CAD/CAM), automation in production, and lean manufacturing. Finally, 16 papers were excluded because either a copy of the full text was not available; the results of the study were reported multiple times, or there was an unclear focus on software development. Hence, a total of 38 publications were left for the subsequent quality assessment and data extraction (see Appendix A).

3.4 Data Extraction and Quality Assessment

We developed a data extraction form on the basis of the research questions posed in Section 3.1. The construction of the form aimed to gather data needed for the data synthesis so that the research questions could be answered. This was primarily done quantitatively by classifying the selected studies into a number of representative properties (systematic mapping)(Petersen et al. 2008), but also qualitatively through in-depth analysis of the data (systematic review) (Kitchenham and Charters 2007). Table 5 describes the properties used in the extraction form and their mapping to the research questions.

Table 5.
Overview of extracted data.

Property	Description	Research question(s)
Main focus	- Lean in large-scale software development - Development phases - Methods and practices used (e.g., Kanban, Kaizen, and concurrent development etc.). - Type of contribution (e.g., model, framework, process, method, practice, metrics etc.).	RQ1
Context	- Domain (e.g., automotive, aerospace, telecommunication, web etc.). - Project type (e.g., size, duration, complexity). - Product type (e.g., system, software application, service complexity).	RQ1
Type of research	Categorization of the research type used in the papers based on the existing classification scheme provided in Wieringa et al. (2006).	RQ2.1
Relevance and rigor	Quality assessment of the papers by evaluating rigor and relevance based on the method used by Ivarsson and Gorschek (2009, 2011).	RQ2.2, RQ2.3
Topic in SE	Mapping of the main contributions of the papers to topic(s) in the SE field according to the KAs in SWEBOK (Abran et al. 2004)	RQ2.4
LPD principles	Mapping of the main contributions of the papers to the LPD principles in the LPDS model (Morgan and Liker 2006).	RQ3

For a deeper analysis, data was enriched by extracting qualitative data including descriptions of the study context and the contributions, reported success factors and limitations, and the objectives of the study.

To mitigate any misinterpretation of the study results due to low study quality, the extraction form also included a quality assessment that was performed for each of the studies. For estimating the study quality, the studies were assigned scores for rigor and relevance according to the evaluation method provided in Ivarsson and Gorschek (2009, 2011).

In order to enhance the precision and consistency of the data extraction, and a common understanding and interpretation of the properties among the researchers, we piloted the data extraction form two times on a limited number of papers. After each pilot, the results were compared and any disagreements between the researchers were resolved through further discussions until consensus was reached. For example, clarifications of the knowledge areas and research types were needed, and it was necessary to divide the level of relationship between the contributions of the papers and the LPD-principles into the three scores of 'no', 'weak' and 'strong' relationship. A 'strong' relationship entails that it is clearly and explicitly stated while a 'weak' relationship means that it is only possible to deduce it.

The piloted data extraction form was then used by one of the researchers, who performed the further data extraction. While doing the data extraction, the form was continuously discussed and updated (e.g., adding, splitting and reformulating categories). The final data extraction form is provided in Appendix B. The extracted data of each paper was filled out and documented in an MS Excel sheet. Furthermore, a short rationale was added explaining why the paper should be in a certain category (e.g., why the paper was attributed to a specific research type). From the entered data, we calculated the frequencies of publications in each category, and used qualitative data coding for scrutinizing and categorizing the extracted quotes (e.g., shared results, claims and recommendations).

4 Results and Analysis

In this section, we present the results of the SMS derived from the 38 papers finally selected according to the research questions specified in Section 3.1. Of these, 16 studies addressed large-scale software development. The 38 included papers (S1-S38) are listed in Appendix A.

4.1 Quality Assessment (RQ2.2 and RQ 2.3)

We performed the quality assessment by evaluating the rigor and relevance of the studies based on the aspects included in the model presented by Ivarsson and Gorschek (2011). The level of the industrial relevance of the reported results is estimated with regard to the realism of the research setting and the applicability of the research method for investigating phenomena in the real world. The level of rigor is estimated through examination of to what extent and detail the context and research method are presented, and the validity of the results is discussed. The maximum score for rigor is three, while relevance has a maximum of four.

Fig. 4 shows the distribution of rigor and relevance of the selected studies, using a bubble graph (Petersen et al. 2008). The size of the bubbles is proportional to the number of papers that are in the pair of scores for relevance and rigor corresponding to the bubble coordinates (the number of papers addressing large-scale software development is placed in brackets). The graph shows that 12 studies have zero relevance (32 percent) and of these nine studies (24 percent) have both zero rigor and relevance where two studies address large-scale software projects. Typically, the 12 studies include descriptions of solutions that were evaluated by using small examples (six out of 12), or relying on the view of the author(s) (four out of 12). However, this does not mean that these studies are out of scope of this review and thus should be excluded. The purpose of the model is to give an overall picture in the field being reviewed by approximating the potential industrial relevance and its progress, rather than providing precise and detailed criteria for an exact classification of each individual study (Ivarsson and Gorschek 2011). Furthermore, our main objective of the SMS is to establish knowledge about state of the art, being helpful for the case companies in the search for solutions to the key issues identified in the SPI project. State of the art deemed as irrelevant according to the model can be relevant for the case companies as there may be other factors influencing the evaluation of the relevance than those based on the actual use in industry. This makes it difficult to distinguish relevant studies from irrelevant ones. For example, the likelihood of introducing subjective bias when assessing the value of research is high in both what to assess (e.g., what constitutes value and quality), and reviewers' competence to assess it (Dybå et al. 2007). Also the time perspective needs to be considered as it takes in the order of 15-20 years before state of the art has matured so it can be implemented in industry (Dybå et al. 2005). However, drawing conclusion merely based on these 12 studies should not be done as the model indicates they have low quality.

On the other hand, 24 studies have relevance equal to three or more than three. Furthermore, 14 out of the 16 studies on large-scale software development have relevance equal to four. The relatively high values of relevance are explained by that many studies are reporting on experiences of applying lean approaches to real industrial settings. However, most of the studies are located in the lower left and right quadrant, where 28 studies have a rigor equal to one or less than one and 11 of these studies are on large-scale software development. Insufficient information about the study design and lacking discussion of the validity are the main reasons for the low values of rigor. For example, 16 studies did not even briefly describe the study design, and only one provided a detailed discussion of the validity. Omitting to report how the study was performed (e.g., sampling, data collection and analysis), where (in what context), and any limitations of the results (e.g., generalizability and reliability) make it difficult to replicate the study and evaluate the results (Dybå et al. 2005; Ivarsson and Gorschek 2011).

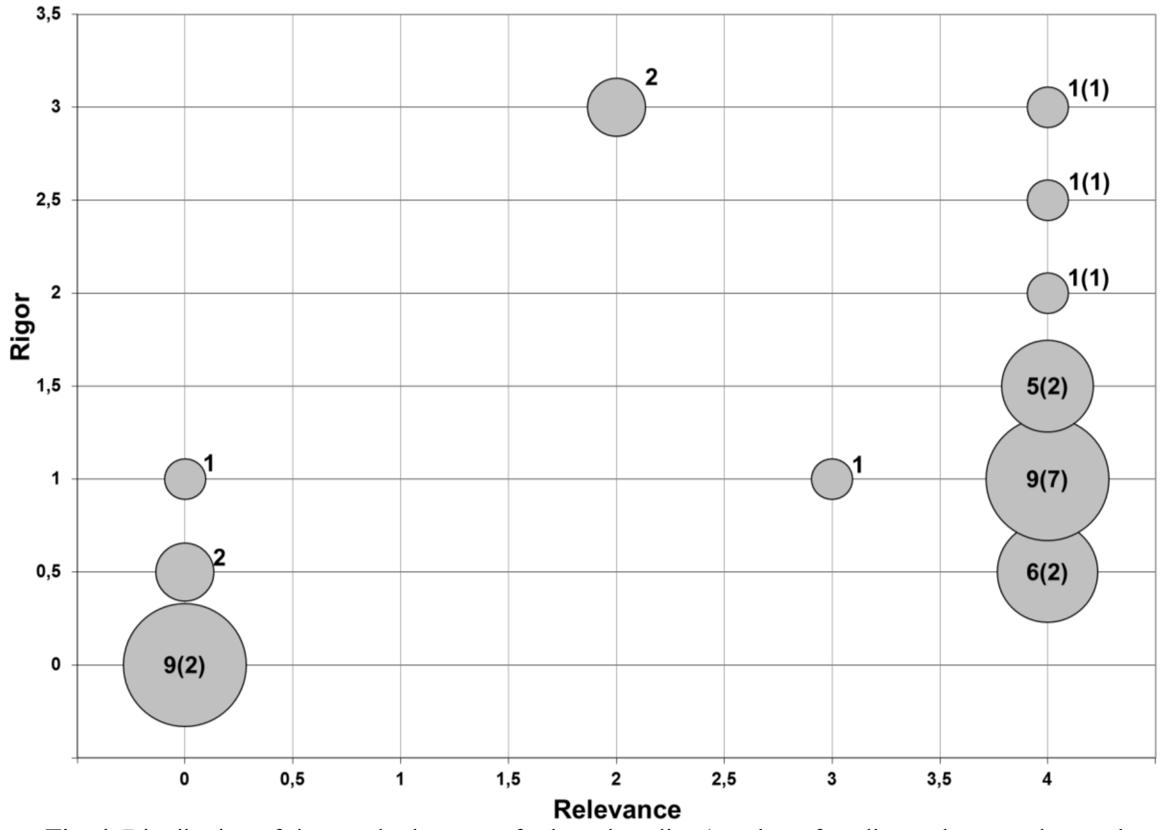

Fig. 4. Distribution of rigor and relevance of selected studies (number of studies on large-scale are placed in brackets).

To summarize, even though the quality assessment shows that the relevance of the studies is relatively high, the lack of rigor reduces the possibility to judge the capability of the state of the art to be contributing for practitioners seeking to adopt new best practices, and limits replications of the studies.

4.2 Current State of the Art (RQ1)

In total 38 studies dealt with state of the art in software development built on lean principles and practices. The studies were performed in contexts ranging from large and complex projects in multi-national companies to student projects in academia. Of these, 42 percent (16 out of 38) clearly dealt with software development in large-scale settings in different industrial sectors, such as aerospace, avionics, telecommunication and industrial automation.

For all the selected studies, we identified and categorized the used lean practices. Each study was then classified into a single category. However, in some studies lean practices were applied in combination with agile methods (hybrids) and in many studies it was unclear which lean or agile method that had been used. Therefore, studies not focusing on specific methods and practices (e.g., studies only mentioning or only implicitly related to a practice) were classified as generic lean or generic lean and agile.

In Fig. 5, the number of studies per category of lean and agile methods is presented. For each category the total number of studies and studies addressing large-scale development are given. For example, the total number of studies classified as generic lean is 14 where nine of these studies are addressing large-scale development. For all the studies, 66 percent (25 out of 38) reported on solely lean, while 34 percent (13 out of 38) reported on lean and agile hybrids. Almost the same relationships can be seen for studies on large-scale development, where 69 percent (11 out of 16) reported on lean and 31 percent (5 out of 16) on hybrids. It is notable that publications reporting on combinations of lean or hybrids of lean and agile, and plan-driven development (e.g., waterfall, RUP, and V-model) could not be found, since such combinations have been recommended for large-scale software development (Boehm 2002; Karlström and Runesson 2005; Sommerville 2007).

Looking at the diagram, most of the studies (24 out of 38) do not focus on any specific lean practice; 14 studies report on generic lean, eight on generic lean and agile, one on generic lean and agile model driven development, and one on generic lean and XP. Pull System (Kanban software development) is the most frequently reported lean practice (six studies), but it can also be observed that this practice has not been reported in large-scale software systems development. Furthermore, 12 out of the 14 studies reporting on a specific lean practice have been published recently (2006-2010). This is much in line with Wang et al.(2012), who saw a trend of adopting more and more concrete lean practices, and in particular, Kanban software development. With regard to studies on large-scale development, 81 percent (13 out of 16) are classified into generic lean (nine studies), generic lean and agile (three studies), and generic lean and agile model driven development (one study). Value stream mapping (two studies) and visual management (one study) are the only specific types of lean practices that are clearly focused on and reported.

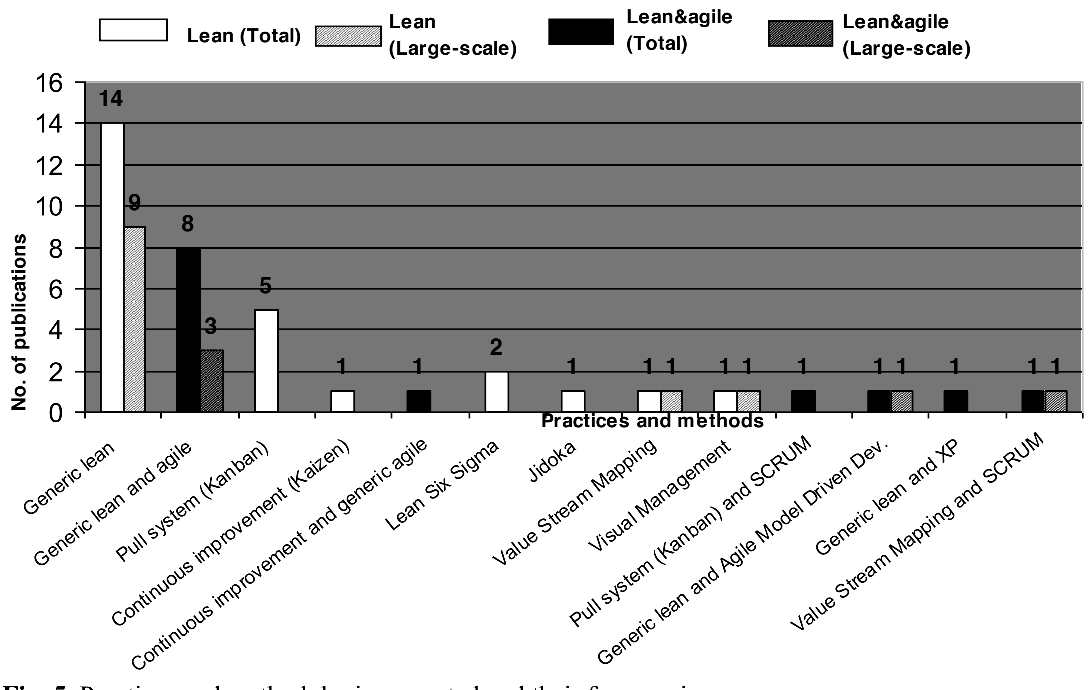

Fig. 5. Practices and methodologies reported and their frequencies.

4.3 Type of Research (RQ2.1)

To obtain an overview of the research type used in the studies, they were classified according to the classes provided in Wieringa et al .(2006). The diagram in Fig. 6 shows the distribution of the research types for all the selected studies (black bars), and the 16 studies on large-scale development (black and white bars). For example, in total 17 studies are classified as experienced studies and of these 10 are studies on large-scale development. It can be observed that the most common research type used is experience papers (45 percent of all studies and 62 percent of studies on large-scale development) in which practitioners have reported their own experiences, followed by evaluation research (21 percent of all studies and 25 percent of the studies on large-scale development). Furthermore, a majority of the publications are non-empirical (76 percent), since only 24 percent (nine out of 38) can be classified as evaluation research and validation research. Of those four studies on large-scale development are empirical. This indicates that the reviewed area in this SMS is not yet mature. One reason for this can be that the lean and agile principles and practices have been introduced relatively recently in the field of SE (e.g., (Beck 2004; Poppendieck and Poppendieck 2003; Schwaber and Beedle 2001). Another reason could be that although LPD has been used for many years by Japanese automakers and Western companies in, for example, the automotive and aerospace industry (Murman 2004; Ward 2007), it is not until recent years that software has become an important component in these traditionally hardware-focused industrial sectors (Venkatech Prasad et al. 2010; Broy et al. 2007). Nevertheless, in order to increase the maturity, there is a need for more validation and evaluation research efforts, involving rigorous research methods (see also Section 4.1). These include, case studies on large-scale software development projects in real industrial settings, where data is collected and analyzed systematically and the validity of the results are scrutinized (Ivarsson and Gorschek 2009, 2011).

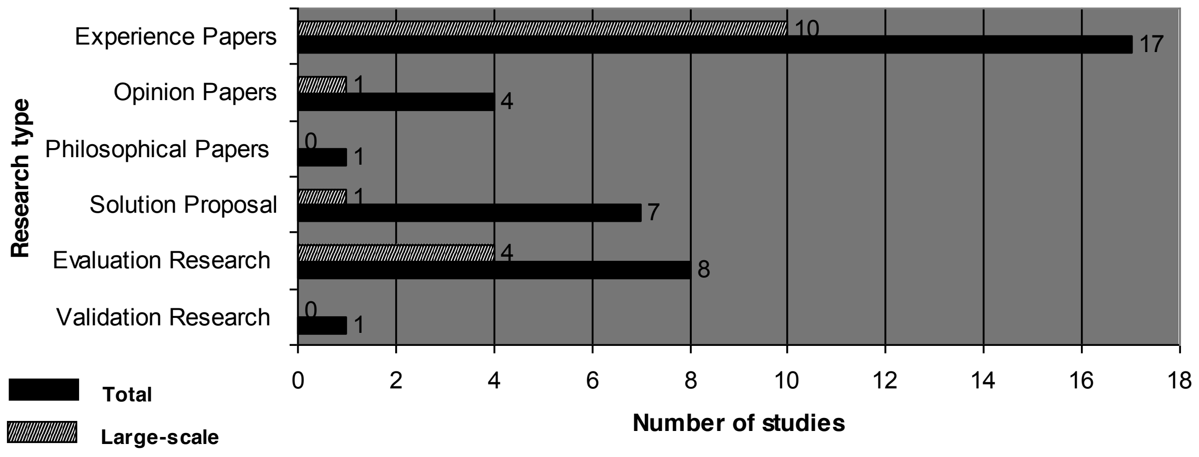

Fig. 6. Research type distribution of all studies (black bars) and studies on large-scale software development (black and white bars).

4.4 Topics in SE (RQ2.4)

We mapped the main contributions of the papers to topic(s) in the SE field according to the KAs in SWEBOK (Abran et al. 2004) in order to obtain an overview of the coverage of the studies in the SE domain. Each study could be mapped to multiple KAs. The diagram in Fig. 7 shows the proportion of all studies (black bars) and studies on large-scale software development (black and white bars) per KA. For example, the proportion of all studies in the KA of engineering tools and methods is 16 percent and the proportion of studies on large scale development in this KA is 10 percent.

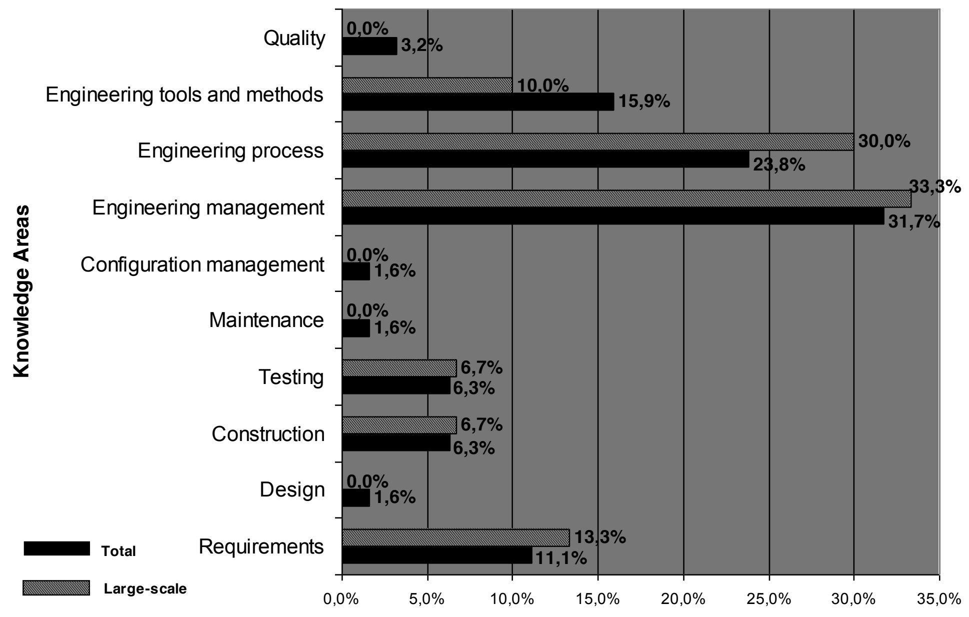

Fig 7. Proportions of studies per KA— all studies (black bars) and studies on large-scale software development (black and white bars).

It can be observed that most of the studies could be classified into the KAs of engineering management (32 percent) and engineering process (24 percent), whereas there are few studies clearly addressing the KAs of configuration management (2 percent), maintenance (2 percent) design (2 percent) and quality (3 percent). The proportions of the studies on large-scale software development show a similar pattern. The largest proportions of these studies are mapped to the KAs of engineering management (30 percent) and engineering process (33 percent), and there are no studies classified into the KAs of configuration management, maintenance, design, and quality. An explanation for this is the broadness of the KAs of engineering management and engineering process spanning over the other KAs, which makes it easier to classify the studies into these two KAs. Many studies that did not clearly specify which KA they addressed, usually dealt with SPI. In order to primarily reducing lead times and enhance quality, these studies involved different ways of tackling issues of general software development management based on lean management principles (e.g., defining customer value and eliminating waste). This indicates that the reviewed area of interest is immature, since most of the studies provide implicit state of the art without detailed and practical guidance for adapting it to different situations and problems within a specific KA. Lacoste (S15) and Linear and Preston (S16), for example, only articulate that Kaizen (continuous improvement) management is a beneficial approach when improving the software life cycle processes. Another example is Perera and Fernando (S24), who conducted an experiment involving ten student projects and found that a hybrid process of lean and agile produces more lines of code than an agile process, but they do not provide a detailed description of how the processes differ and which practices are used.

Along with the lack of rigor, this makes it difficult for practitioners to understand the applicability of the state of the art reported. For example, four studies on large-scale software development could be mapped to the KA of requirements and of these only two had a rigor of more than one, namely Ippolito and Murman (S11) and Petersen and Wohlin (S25). Ippolito and Murman (S11) conducted three detailed case studies and 128 surveys. With the goal of improving the software upgrade value stream in development of large-scale software-intensive systems, they sought to identify effective lean practices for eliciting software requirements from aerospace system level requirements. However, the reported findings and recommendations only formed a high-level basis for developing a framework that would increase the value-added contribution of the software requirement process. Petersen and Wohlin (S25), on the other hand, provide examples of actual uses of a method influenced by the Quality Improvement Paradigm (QIP) (Basili 1985) called Software Process Improvement through Lean Measurement (SPI-LEAM). SPI-LEAM was designed for improving the flow in software development and applied in large-scale industrial setting for measuring inventories of software requirements.

4.5 Relationships between the State of the Art and LPD Principles (RQ3)

To answer RQ3, we identified relationships between the main contributions of the selected studies and the principles in LPDS. Each study could have multiple relationships. Table 6 describes the principles, lists all the selected studies, and shows the number of them for each principle. Relationships to studies on large-scale software development are also shown and their frequencies are placed in brackets. Furthermore, only the strong relationships identified (see Section 3.4) are given in Table 6 (Appendix C shows both the strong and weak relationships). For example, there are nine studies with strong relationships to Principle 1, and of these four studies (S11, S13, S20, and S27) are on large-scale software projects.

Looking at Table 6 and how the strong relationships identified for all studies were scattered over the sub-systems in LPDS, 42 relationships could be attributed to process, 15 to skilled people and 11 to tools and technology. The number of relationships between studies on large-scale development and the sub-systems are 16 for process, seven for skilled people, and six for tools and technology. Thus, a majority of the studies have a process-oriented focus. Even though the importance of skilled people has been acknowledge as a major factor for success in software projects (Brooks 1987; Kraut and Streeter 1995), this indicates that there is still a strong belief that the processes must be changed for effectively producing software-intensive systems.

Both for all studies and those on large-scale software development, Principle 3 has the highest number of relationships (25 and 11 respectively), followed by Principle 12 (10 and five respectively) and Principle 1 (nine and four respectively). Apparently, the main areas of interest in the studies are lean principles and practices for defining customer value, reducing waste, creating flow in the PD process, and visual management. Moreover, none of the studies have a strong relationship to Principles 5, 8 and 13, which indicates an overall lack of research on the role of product manager, integration of suppliers, and tools for standardization and organizational learning in lean approaches to development of software. It can also be seen that there are no strong relationships between Principle 10 and studies on large-scale software development.

In the following, this section presents a deeper analysis of the strong relationships between the principles and the studies on large-scale software development. Thus, Principles 5, 8, 13 and 10 are not further analyzed. In order to assist the case companies to evaluate the potential value of the results prior to deciding on adopting the state of the art, practical implications are analyzed and reflected upon based on the data extracted from the reviewed papers and on previous observations in the case study by Pernstal et al.(2012). In addition, potential gaps and implications for future research are discussed.

Table 6
Relationships between lean principles in LPDS (Morgan and Liker 2006) and selected studies

Sub-system	Principle	Description	Strong relationship		
			Total	Large-scale	Freq.
Process	1.Establish customer-defined value to separate value-added activity from waste.	To create a lean PD process, it is important to establish a customer defined value as a first step. Once these values have been identified, diffused and understood throughout the whole organization, it is possible eliminate waste.	S7,S11,S13, S14 S20,S21, S26,S27 S33	S11, S13, S20,S27	9 (4)
	2. Front-load the PD process while there is maximum design space to explore alternative thoroughly.	Involves problem-solving at root cause level in early project phases. The aim is to eliminate late engineering changes like 'quick fixes' and patches that rarely result in increased product or process performance.	S7,S8,S12, S18,S26,S27, S33	S27	7 (1)
	3. Create a leveled PD process flow.	Involves eliminating waste (everything that does not contribute to the value for the customer) and establishing flow (regular pace) in the PD process. The total PD value stream is examined with the aim to eliminate non-value adding activities that occur between development steps such as unnecessary handovers of documents and reinvention instead of standardization of components. Flow is created by incremental development where work is broken down into suitable tasks.	S3,S6,S7,S8, S9,S10,S11, S12,S13,S14, S15,S17,S18, S19,S20,S22, S23,S25,S28, S29,S30,S32, S34,S37,S38	S3, S11, S13, S20, S22, S23, S25, S28, S30, S32, S37	25 (11)
	4. Utilize rigorous standardization to reduce variation, and create flexibility and predictable outcomes.	Standardization has a large influence on PD since it reduces variation that enables increased flexibility and predictable outcomes. There are three categories of standardization: design standardization, process standardization, and engineering skill-set standardization.	S32	S32	1 (1)
Skilled People	5. Develop a chief engineer system to integrate development from start to finish.	The top management appoints a chief engineer immediately after a new program has been decided upon. The chief engineer is considered to be the owner of the product and is responsible for the whole development process from concepts to launch.			0
	6. Organize to balance functional expertise and cross-functional integration.	Creating efficient PD organizations by combining the benefits of product and functional focused structures in a matrix organization. This allows simultaneous attention to functional and program demands.	S36	S36	1 (1)
	7. Develop towering technical competence in all engineers.	The necessity to use a rigorous recruitment process, mentoring and on-the-job-training (OJT) in a structured way. For example, in order to have the capability to technically challenge design engineers, the recruitment and training of manufacturing engineers should be equally comprehensive.	S11,S16,S23, S29	S11, S23	4 (2)
	8. Fully integrate suppliers into the Product Development System.	Comprises a high degree of supplier involvement. This implies early involvement of suppliers in PD, a rigorous selection of suppliers, and that suppliers are committed to continuously maintain and develop their engineering and manufacturing capabilities in order to meet the demands of the ordering company, i.e. original engineering manufacturer (OEM).			0
	9. Build in learning and continuous improvement.	To achieve continuous improvement (Kaizen) of products or processes, it is important to recognize and encourage learning and understanding of technologies and processes where both tacit and explicit knowledge are developed, diffused and maintained in the organization n.	S3,S13,S15, S16,S22,S26, S27,S35	S3, S13, S22, S27	8 (4)
	10. Build a culture to support excellence and relentless improvement.	The culture embraces a fairly stable set of assumptions that are taken-for-granted, shared beliefs, meanings, and values in an organization that govern the members' operations and enables the organization to rely less on formal lean control systems.	S16, S19		2 (0)
Tools & Technology	11. Adapt technology to fit your people and processes.	Tools and technology must be customized based on organizational needs. This means that the integration of new technologies facilitating incorporation with existing systems or tools and adaptation to established processes should be seamless, and not vice versa.	S31	S31	1 (1)
	12. Align your organization through simple visual communication	Deals with the organization's capability of effectively coordinating complex communication such as requirements, test results, project status reports and manufacturing constraints between teams and across functions in the PD process.	S6,S8,S11, S14,S19,S20 S25,S28,S31, S38	S11, S20, S25, S28, S31	10 (5)
	13. Use powerful tools for standardization and organizational learning	To build learning organizations, it is necessary to deploy tools that support development, diffusion and preservation of both explicit and tacit knowledge, on which evolving standards are based.			0

Principle 1—Establish Customer Defined Value: To suppress the occurrence of waste, lean organizations emphasize elicitation and dissemination of defined customer values throughout the organization to all involved teams by breaking down the overall goals to meaningful objectives on all levels of the organization. This leads to increased understanding in downstream activities of what creates value for the customer.

Kettunen (S13) presents a research model for investigating and evaluating the performance of process improvements based on lean and agile principles and practices in software development organizations. Specifying customer value is central in the model as the performance on enterprise level is expected to be based on continuous delivery of high product value. In a case study by Middleton et al. (S20), they claim that customer value is the most important issue to initially focus on in software development. To define customer value, the development team they studied, therefore iteratively elicited and refined requirements in close cooperation with the customer, yielding a list of prioritized features. When prioritizing the requirements, the Kano model (Kano 1996) describing the relationship between customer satisfaction and quality and categorizing customer needs into three types, was found useful. However, Ippolito and Murman (S11) found that eliciting the customer value of software changes in large systems is complex as they are dependent upon multiple, interacting values associated with other changes to the whole system (e.g., sensors, certifications, hardware, and supporting equipment). Consequently, this creates multiple value streams where the development of software includes just one part of identified customer values of the complete set of values for developing the total system.
Dissemination of defined customer values throughout the organization to all involved teams leads to increased understanding in downstream activities of what creates value for the customer. Middleton et al. (S20) and Rudolf and Paulish (S27) report on using design structure matrix (DSM) (Eppinger et al. 1994)for breaking down defined customer requirements in order to transform them into design requirements that are deployed to further the PD process, and ensure value-adding requirements on lower abstraction levels.
All four studies on large-scale development mapped to this principle address how customer value is identified and broken down for the design of software-intensive systems, but not for other functions of a development organization. Referring to our case, there are no studies on how to transfer customer value to Man and breaking it down to best practices that create most value for the customer of these systems in the manufacturing operations. For example, one observation was difficulties in determining the types of software-intensive systems in vehicles that are most beneficial to configure in the manufacturing processes and how this should be performed effectively (e.g., configuration by assembly plant software download or parameter setting of pre-loaded software). Thus, for a better understanding of how to create customer value concerning design and manufacturing aspects of large software-intensive systems, there is a need of more studies, which in turn can provide specific knowledge and best practices that can be added to LPD for large-scale development of software-intensive systems.

Principle 2—Front-load the PD Process: Front-loading encompasses clarifications and trade-offs of different aspects and requirements in early phases of the PD process in order to obtain more robust and optimized systems. This reduces the risk for rework in late development phases. Rudolf and Paulish (S27) was the only study on large-scale software development that had a strong relationship to this principle. To achieve early problem-solving, they used root cause analysis based on A3s and 5Whys. However, the study merely report on what to do on a high level, but not how the actual implementation should look like. Furthermore, evidence for the applicability of the presented state of the art is unclear as Rudolf and Paulish (S27) only give brief descriptions and experiences of applying problem-solving in the case company.
Active involvement of manufacturing engineers in early phases is recognized in research as one of the most critical factors in PD, since it reduces the risk of pushing through manufacturing

prerequisites in late phases, which often leads to costly changes and jeopardizes the launch of the product (Sobek et al. 1999; Wheelwright and Clark 1994).

Difficulties in attaining early and pro-active manufacturing involvement was one of the main issues identified at the case companies. The primary reasons for this were that software-intensive systems are intangible and are often described in written requirements specifications implying a high abstraction level to be interpreted and understood. In addition, the manufacturing processes and tools that are affected by in-vehicle software are often recognized as complex, since they incorporate both vehicle communication technologies and interaction with other IT systems that integrate PD and Man (such as product data management systems and factory systems). Being able to understand and foresee the impact of software-intensive systems on the manufacturing processes in early phases is therefore often highly dependent on deep knowledge and a great deal of experience among both manufacturing and design engineers. For example, designing the audio systems in a vehicle without foreseeing required failure diagnostics for the loudspeakers and buttons, can make it difficult and costly to secure the quality of the system in the manufacturing processes (for example, manual intervention is needed). Integrated problem-solving (Wheelwright and Clark 1994) and Set Based Concurrent Engineering (SBCE) (Sobek et al. 1999) are lean practices that elevate active cross-departmental development work, but they do not consider specific contextual needs. Thus, there is a need for future research giving a better understanding of early and active communication and balancing of demands on products and manufacturing operations in large-scale development of software-intensive systems, on which specific methods and practices can be developed and effectively applied, and suggested as add-ons to this LPD principle.

Principle 3—Eliminate Waste and Create Flow: Identifying and eliminating sources of waste in a value stream by using value stream mapping (VSM) (Womack and Jones 2003) are central in Principle 3. When using VSM in PD, the available and required resources from a holistic view of the PD system are mapped out, enabling increased understanding of waste and the sources of waste in a PD value stream (Morgan and Liker 2006; Poppendieck and Poppendieck 2003). Based on the seven wastes of manufacturing (Ohno 1988), Morgan and Liker (2006) provide corresponding sources of waste in PD, and Poppendieck and Poppendieck (2003) specifically interpret them to suit the SE domain.

For a considerable amount (25, equaling 66 percent) of the total 38 studies, we could identify a strong relationship to Principle 3. Of these, eleven studies addressed large-scale software development where six of them dealt with elimination of waste. Table 7 gives an overview of what type of waste each of these six studies focused on, and gives a description of each waste in PD and examples related to SE.

Table 7
Seven wastes

Seven wastes	Description	Studies on large-scale
1. *Overproduction*—producing more or earlier than the next process needs.	* When completing design tasks before the next development step in the PD value stream flow, there is a need to process the tasks (e.g., finish coding before testing) * Carrying out unnecessary activities that are not required for the next step or developing extra features without customer value (e.g., developing software for functionality that does not provide value to the customer).	S28
2. *Waiting*—waiting for materials, information, or decisions.	Development engineers often have to wait for reviews, decisions, permissions or information before they can perform dedicated key activities in the PD process (e.g., handshaking and sign-offs of software requirements specifications).	S22, S28
3. *Conveyance*—moving material or information from place to place.	Making unnecessary transfers between activities such as exchange of information between actors and diffusion of decisions throughout the development team leads to loss of momentum, information and accountability in the PD process (e.g., many handovers of written code and results of system integration or verification).	S28
4. *Processing*—doing unnecessary processing on tasks or an unnecessary task.	Unnecessary or incorrect engineering, such as designing from scratch instead of using carry-over or standardized components (e.g., refactoring of working code is omitted), or developing of unique manufacturing processes to fit a specific system or vehicle program instead of striving for standards and commonality.	S20, S22, S28
5. *Inventory*—a build-up of material or information that is not being used.	*Information waiting in queues to be processed by the next step where information gets lost (e.g., designers cannot manage large batches of approved software requirements specifications, creating a risk of omitting important information). *Tasks that are partially done or transferred too late to where it is needed (e.g., code is written, but not integrated in the system).	S3, S20, S25, S28
6. *Motion*—excess motion or activity during task execution.	*Development engineers may attend unnecessary meetings (e.g., redundant review meetings for status reports on coding, system integration or test results). *Unnecessary distances between program members (e.g., suppliers, designers and manufacturing engineers, may create inefficient transfers of information or knowledge, leading to inappropriate design decisions).	S20, S22, S23, S28
7. *Correction*—fixing problems.	*Any additional work for fixing problems that could have been prevented earlier in the product development (e.g., late and expensive in-vehicle software changes due to insufficient adaptation to manufacturing processes affected by the software or occurrence of bugs). Thus, planned development activities, such as Poka-yoke (e.g., checklists, standards, and detailed test plans), for catching and preventing errors in product development, cannot be directly referred to this type of waste.	S20, S28

Table 7 shows that four studies can be attributed to the waste caused by inventories. An underlying reason for this may be the well-known just-in-time approach adopted in manufacturing aiming to eliminate inventories, and that research on general PD recognizes that inventory influences other types of waste negatively (Morgan and Liker 2006). For example, Middleton et al. (S20) claim that minimizing inventories of unverified code and large volumes of requirements fosters the development teams to uncover hidden defects earlier in the PD flow, which reduces rework in late phases. Peterson and Wohlin (S25) claim that inventories also increase waiting times, the number of developed extra features without customer value, and uncompleted tasks. For measuring different types of inventories in software development, they present the SPI-LEAM method. SPI-LEAM was applied to a real case, where the inventory of requirements were measured on the overall process life-cycle. This led to a number of suggested improvements. For example, to deal with overload situations in the development team, the team pulled prioritized requirements from a buffer instead of pushing requirements into development.

The only study that could be related to all types of waste was Sekimura and Maruyama (S28), who introduced and applied TPS to large-scale development of business application software. Instead of focusing on one type of waste, they emphasize the importance of identifying and eliminating all seven types of waste. For example, work allotments were changed for reducing waste in processing, recurrence of defects were prevented by analyzing and taking corrective actions based on software defect reports, and inventories were eliminated in terms of answer rates within 24 hours.

Although VSM is a central practice in lean, only two studies report on using VSM in developing software-intensive systems. In a case study by Mujtuba et al. (S22), VSM was applied to a software product customization process. On the basis of a static validation they conclude that VSM was useful and its benefits can be conveyed to more general SPI efforts. They identified waste related to waiting (e.g., delays for customer sign-offs and system integration), processes (e.g., extra processes for design) and motion (e.g., motion of requirements). Similarly, Pernell-Klabo (S23) experienced that VSM is a helpful practice when introducing and transitioning towards agile software development where motion was the most prevalent type of waste identified, followed by processing and overproduction.

In addition to eliminating waste, lean organizations endeavor to establish a flow (i.e. regular pace) of material and/or information. In software development, software concepts of object-oriented programming and modular design stress the need to decompose the work packages into the smallest units possible, because there is a complexity explosion and the chances of having a defect and the cost of finding it goes up exponentially with the size of the build (Hoffman and Weiss 2001).

Accordingly, to establish and maintain flow, most of the studies suggest incremental development where previously large project tasks are divided into smaller chunks (Aoyama (S3); Middleton (S20); Sutton et al. (S32); Vodde (S37)). For example, Aoyama (S3) introduces and applies a model for managing development of large-scale software systems in a concurrent manner, named concurrent development system (CDS). CDS is the ancestor to the agile software process model (ASP) (Aoyama 1997). The tasks were divided into so-called minor and major enhancements depending on the possibilities to decouple the parts of the systems developed. To facilitate release planning and enable a smooth development flow, the development time of major enhancements was set to a multiple of minor enhancement. In this study, the major enhancements were six months and the minor three months. Furthermore, the development process was divided into an upper sub-process including requirements analysis, design, and implementation, and a lower sub-process involving integration and system tests. Each sub-process had to be completed within a fixed cycle time (here three months) with a fixed number of developers, and iterated over multiple releases (cf. sprints in Scrum (Schwaber and Beedle 2001)). Applying the CDS resulted in a shorter development cycle time (from one year to three months), and reduced fluctuations in development sizes and build-up of inventories, improving the utilization ratio of workload. Similarly, Middleton et al. (S20) balance the workload by breaking down projects into manageable chunks of work, termed 'kits' (batches of requirements are decomposed into 'stories'), which in turn enable the setting of a cycle time. Furthermore, they observed that staff with multiple skills has a positive impact on reducing fluctuations and in workload.

Overall, when creating flow and eliminating waste in large scale software development, there is little research and evidence of the benefits and limitations of lean practices or tools. For example, different types of waste seem to occur depending on the degree of exploration and focus in the research. This indicates that solely focusing on one certain type of waste, may imply a risk of factors outside this waste area being left out of the analysis in the efforts of eliminating waste. For example, despite that it was well motivated to develop a method for measuring inventories in Peterson and Wohlin (S25), it can be questioned whether inventories is the most influencing and prevalent type of waste in a software development flow.

An observation in the case study was that the types of waste in the PD and Man interface are multifaceted. For example, most software-related issues (e.g., inadequate vehicle diagnostics implementation) come to light in the pre-production evaluation phase peaking somewhere in the

middle of this stage. Consequently, this causes unplanned and costly design loop backs of both in-vehicle software and affected manufacturing processes in the final part of the project.
Another example is managing the large interface between PD and Man usually also entails a comprehensive meeting structure causing unnecessary meetings and overwhelming inter-departmental transferring of information. In addition, the development of increasingly software-intensive automotive systems creates more complex interdependencies between previously decoupled and modular systems in vehicles (e.g., the CLS system as described in Section 2.1). This makes it even more important to ensure cross departmental information transfer whilst not inflating the meeting structure and efforts to deliver information. It is, however, rarely an option to dedicate more resources for integrating these systems and the manufacturing processes in highly competitive businesses, such as automotives. This will probably also diminishing the true needs of interaction and thereby hinder the possibilities to uncover and reduce waste.

Principle 4—Utilize a Rigorous Standardization: Standardization is important as it reduces variation, which enables increased responsiveness to change and flexibility and predictable outcome. Morgan and Liker (2006) classify standardization into three categories, design, process, and engineering skill set. However, reducing variability in PD is complicated as there are good and bad variability (Reinertsen 2009). Good variability adds value and should be exploited while bad variability (e.g., sloppiness and repeating mistakes) should be eliminated, but it is often difficult to distinguish between them. Consequently, to accomplish the required response in PD to rapid changes without reducing productivity and system quality, it is necessary to pursue standardization of processes and design and to achieve process discipline among employees without suppressing variability that increases an organization's success.
One of the future key challenges in SE is that development of software must be responsive to rapid changes without compromising system quality (Sommerville 2007). However, only one of the large-scale studies is concerned with standardization in lean development. For spreading lean throughout the aerospace industry, Sutton (S32) reports on the experiences from using the LAI program (Murman 2004), which was initiated by the U. S. Air Force in 1992. Responsiveness to change is one of the meta-principles in LAI, emphasizing standardization through domain engineering. This implies that requirements and architecture are structured along domain lines, which provide a standardized and stable basis for variants. The LAI practices and principles were applied to a software project, and overall, Sutton concluded that it was a success as it reduced cost and risk. However, the impact of standardization is not explicitly discussed.
Reflecting on this principle in relation to our case, the software-intensive systems entail an increase of variants being dealt with in the automotive industry. A premium car typically has about 80 electronic fittings that interact on several networks in the car, and can be ordered depending on, for example, the country etc.(Broy et al. 2007). Simple yes or no decisions for each function yield a possible maximum of roughly 2^{80} variants to be ordered and produced for a car. In addition, differences between production units in terms of available processes, assembly sequences, and tools lead to excessive work for adapting the variabilities in design and manufacturing. For example, adapting variants of networks (e.g., CAN and FlexRay), and diagnostics and software download concepts (e.g., ISO 14229-1 2006) in the vehicles, and different tools for communicating with the vehicles in manufacturing, increase the complexity and effort of managing software-intensive systems in production. Although partnerships such as Automotive Open System Architecture (Autosar 2009) aim to standardize software-intensive systems in the automotive domain, the SMS presented in this paper shows that there is a lack of research providing better

understanding and advice, and helping the standardization of design, processes and competences across engineering disciplines and departments in the context of large-scale software-intensive systems development.

Principle 7—Develop Towering Technical Competence: This concerns the importance of having comprehensive recruitment, structured and sophisticated training, and mentoring programs in order to obtain necessary skills needed to perform ones work, but also sufficient understanding of others work.

Two of the studies addressing large-scale software development are concerned with Principle 7. When introducing and changing from traditional to lean approaches in software development, Pernell-Klabo (S23) experienced the importance of informing and training the staff. After the use of training workshops and pilot projects, the staff became more motivated for change, and positive results were achieved. For example, 40 percent of the lead time was reduced. Ippolito and Murman (S11) found that the survey respondents believed that on-the-job training (OJT) is the most widespread method and the only effective training method while formal training is less effective. This view is emphasized in lean companies, which are aware of the necessity of OJT as new engineers can develop their own working procedures if they are thrown into projects without proper guidance, leading to increased variability in the product development system (Morgan and Liker 2006).

This principle relates to one issue found in the case study showing that there was a lack of mutual understanding of the work done by design and manufacturing engineers. For example, the knowledge and experience of manufacturing operations affected by in-vehicle software among design engineers, and the level of SE competence within manufacturing. In related earlier work, low understanding of each other's work has been found as a major cause for gaps in software requirements communication (Bjarnarsson et al. 2009), but also a critical factor in the PD and Man interface for a successful production start (Lakemond et al. 2007; Vandevelde andVan Dierdonk 2003). This can be improved by encouraging the staff to experience different functions (e.g., job rotation), and comprehensive and thorough recruitment of, for example, manufacturing engineers in order to give them a better capability to understand and technically challenge design engineers (Carlsson 1991; Morgan and Liker 2006; Nihtilä 1999).

Developing necessary skills and mutual understanding of each other's work across departments in large-scale software development seems to be important, but the results of the SMS presented here shows that research addressing this LPD principle is scarce.

Principle 9—Build in learning and Continuous Improvement: Establishing, maintaining, and capitalizing on continuous improvement are dependent on organizations' capability of building learning organizations where development, diffusion, and maintenance of organizational know-how, are natural tasks in daily work (Takeuchi and Nonaka 1986). Similarly, for SPI there are software process maturity models such as the capability maturity model integration (CMMI) (CMMI 2010) and Automotive SPICE (Automotive SIG. 2010). However, these are high level frameworks that do not detail how the actual implementation should look like in the actual industrial setting as they adopt one-size-fits-all view across companies and projects (Fayad and Laitnen 1997; Kuilboer and Ashrafi 2000; Zahran 1998).

Four of the studies on large-scale software development pay attention to Principle 9. To accomplish continuous improvements across a company, Rudolf and Paulish (S27) established a cross-business unit community. The community has regular meetings where general improvement frameworks and practices are shared and refined, so they can be adopted and standardized across the company. Furthermore, the improvements must be based on an understanding of the actual situation, which is best obtained by engaging people close to the identified problems and actually doing the work. Similarly, Aoyama (S3) established a central project SE team (PSET) coordinating feedback from developers and supporting management for achieving continuous improvement of the development processes. In addition, Kettunen (S13) suggests that lean-related process improvements do not only involve learning from incremental SPI, but also learning from more radical shifts of the business processes, which can be managed by organizational development programs.

In the case study we found that there are particular difficulties in specifying manufacturing requirements to be understandable and convertible to measurable parameters for developers of software-intensive systems. In line with Almefelt et al. (2006), the main reason for this was that they were often experienced-based rather than being specifications of purposes and goals, and that they describe expected results (i.e. tacit knowledge). For example, the designs of software-intensive systems are expected to allow effective configuration and quality assurance of the systems in manufacturing. In large-scale software projects, Kraut and Streeter (1995) stress the value of combining both formal communication (e.g., written and transferred specifications and structured meetings) and informal communication (e.g., unscheduled face-to-face meetings and e-mail or phone conversations) across organizational boundaries. Furthermore, it is well-known that much information in software development is tacit and never written down (1995). Consequently, only focusing on improving the quality of the Man and PD specifications of explicit know-how and the formal processes for transferring them, is not enough in order to effectively accomplish continuous improvement.

Even though the LPD principle of continuous improvement has been widely promoted and adopted in industry, there are very few studies on how organizations developing large-scale software manage to accomplish this, and evidence of the benefits and limitations in such organizations is unclear.

Principle 11—Adapt Technology to Fit People and Processes: This principle concerns the necessity of customizing tools and technology to achieve LPD. For this, five sub-principles are given in LPDS. They are presented in Table 8.

Table 8
Sub-principles of Principle 11.

Sub-principle	Description
1. Technologies must be seamlessly integrated.	Introducing and implementing new technologies must allow a smooth integration of them and existing systems and technologies.
2. Technologies should support the process — not drive it.	New technologies must be adopted to established processes and not vice versa.
3. Technologies should enhance people—not replace them.	Instead of motivating investments in new tools and technology by downsizing, it is more important to value and use the personnel's technical experience, skills, and expertise.
4. Specific solution oriented—not a silver bullet.	There are no magic tools or methods that can replace hard work by skilled personnel.
5. The right size—not king size .	Technologies providing the best performance are rarely the ones that are most effective for improving PD.

Staron et al. (S31) was the only study on large-scale software development that had a strong relationship to Principle 11. They conducted an action research project in close cooperation with industry, where prediction methods for forecasting a defect backlog in large streamline software development projects were developed and evaluated. When assessing different prediction models, an important finding in their study was that too complex methods are very vulnerable to changes in the ways of working at companies, which means that they are reluctant to adopt them. This is in line with Weber and Weisbrod (2003), who report on challenges and experiences of RE in development of software-intensive automotives systems. They particularly emphasize the necessity of developing adaptable tools and methods that support engineers in their daily tasks, since there is otherwise a significant risk of users rejecting the tools. For example, they experienced that introduced tools and methods for consolidating RE and model-based development (MBD) were often discarded after a while after being used in projects.

In the case study, both case companies expressed a need of MBD tools for enabling earlier and more frequent prototyping loops of automotive software-intensive systems, which can support early verification of manufacturing operations affected by these systems. Like today's virtual build events (VBE) for securing the assembly of mechanical parts, the goal is to enable the export of digital models of software-intensive automotive systems from the systems used for MBD to the manufacturing systems, or vice versa.

When developing such MBD technologies, the five sub-principles of Principle 11 should be considered. For example, manufacturing engineers should be involved in the further development of modeling tools and working procedures, and it is preferred that this development is an evolving process rather than a big bang. In effect, it may start with an elaboration of work procedures comprising analyses based on observations of computerized visualizations of the software-intensive systems on an appropriate abstraction level (e.g., the functional level), and by using cross-functional techniques such as FMEA together with checklists and requirements containing manufacturing prerequisites. In the end, the tools and methods allow fully automized VBEs, where the complete vehicle can be modeled and validated in virtual representations of the manufacturing processes.

However, available know-how and support in LPD and earlier work for actually using the sub-principles when adopting new tools and technologies for large scale software development, seems to be very limited.

Principle 12—Align the Organization Through Simple Visual Communication: When reducing lead-times in PD by replacing traditional sequential over-the-wall processes with concurrent development practices, studies on PD reveal that communication becomes one of the most critical factors in PD (e.g., (Clark and Fujimoto 1991). In particular, communication and coordination of workgroups that develop interdependent pieces of large software system, is crucial for a successful outcome of the development effort (Kraut and Streeter 1996).

Overall, many introductions of LPD typically start with visualization (planning, action and status boards), and one reason for this may be the desire of management to strengthen and maintain their control when LPD is deployed (Holmdal 2010). Thus, it is a little surprising that a majority of the studies do not map to this principle.

Five studies on large-scale software development are mapped to Principle 12. Two of them report on visualizing the state of the software project, using boards divided into different areas with the aim to uncover project abnormalities (e.g., time and defects). Middleton et al. (S20) focus on continuously monitoring the velocity of the team, and they retrospectively improve the accuracy of estimated time and resources by posting the number of units of work completed over time compared to a target. Sekimura and Maruyama (S28) visualized project management information on an electronic board as the development was performed at several locations and in parallel. For ensuring that correct information were used, the personnel had to follow three rules (1) daily submission of work time, (2) save information in specified folders on shared server, and (3) correctly describe information on prescribed worksheets. However, Ippolito and Murman (S11) found that the methods for measuring and indicating project properties are often deficient. For example, their results showed that the end-to-end cycle time is not well understood by the process leadership leading to difficulties in making product cost and performance design trades, with respect to total time.

Staron et al. (S31) and Peterson and Wohlin (S25) (SPI-LEAM) were the only studies suggesting methods for measuring and presenting project metrics. Staron et al. (S31) present a metric that predicts the number of defects in large-scale software projects using a mix of lean and agile principles that will be open during a particular week. The metric is based on moving average combined with the current level of a defect backlog. For visualizing and communicating the forecast of a defect trend to stakeholders, they simplified the metric and packeted it as an indicator (an arrow) into an MS Vista Gadget. In SPI-LEAM, the inventories were measured as the number of requirements and displayed in cumulative flow diagrams visualizing undesired behavior of the development (e.g., bottlenecks). The flexibility of SPI-LEAM allows it to be used for different types of development artifacts that can be selected to fit specific organizational needs.

As mentioned above, one of the issues found in the case study concerned problems in managing specifications between PD and Man, and in particular, requirements. For improving this, SPI-LEAM may be used to measure the number of requirements that have been transferred from PD to Man, or vice versa, and are waiting to be understood and handshaked by the counterpart. However, the inventory measurements only indicate the level of requirements, but the produced level of quality of the requirements is not shown. For example, to what extent are the manufacturing requirements understood by design engineers? Therefore, it is also necessary to use methods that aim to ensure that the development processes generate the right quality of the requirements. One example is Fricker et al.(2010), who report on a method for improving software requirements negotiation.

4.6 Validity threats

An overall challenge in this SMS was to define the scope, since the investigated area is multidisciplinary and spans the fields of, for example, SE, systems and manufacturing engineering, and management. Searches across disciplines are difficult, as different terminology for the same notion is often used and must be dealt with when defining the search criteria (e.g., electronic and embedded systems in systems engineering and software development in SE). Since LPD has started to be widely adopted in many industries developing complex software systems in large organizations, and communication and coordination across departments is critical for such development (automotive and aerospace), we have chosen focus on lean principles and practices. Although the investigation takes a broad view covering different engineering disciplines and industrial sectors developing software, and was not restricted to empirical research, this is a limitation of our results.

One of the major threats to this study is selection bias. As a protection against this threat, we used three main strategies. First, to balance the comprehensiveness and precision of the search string, we performed several trial searches in the Scopus database, where we tested alternative keywords and combinations of them, see Section 3.2. Furthermore, to capture most of the relevant studies, the publication year was set to be between 1990 and 2010, since the lean paradigm was first introduced by Womack et al. (1990) in 1990. Second, we collected publications from different sources including the ACM Digital Library, IEEE Xplore, Inspec, ISI Web of Science, and Scopus. To ensure we located all studies published in 2010, the final search was performed in February

2011. Third, to guard against built-in bias of the selection based on screening of title, keywords and abstract, the criteria for inclusion and the screening process was elaborated by three researchers. The screening was piloted twice on randomly selected papers, and the consistency was evaluated by calculating the Fleiss Kappa value (Fleiss 1971). The first pilot involved 100 papers and resulted in a too low inter-rater agreement (0.5). Therefore, the inclusion criteria and the screening process were refactored (e.g., better clarifications of the inclusion criteria) and a second pilot was performed. The inter-rater agreement increased to 0.76, which was deemed acceptable based on the recommendations provided in Landis and Koch (1977).

However, basing the inclusion of publications on reading the abstract is a limitation of this study, since the abstract may not reflect the content of the paper and relevant papers can have been missed. In order to mitigate this threat, the inclusion criteria can be tested on a number of papers that are randomly selected for full text reading (e.g., ten percent of the papers). Alternatively, the screening could have been performed in parallel by several researchers, enabling triangulation of the results. However, due to the large number of publication that were left for the screening (10,230), it was deemed not feasible to perform any of these alternatives with the available resources.

Data extraction bias is another major threat to this study. Usually, the studies are possible to classify into multiple classes (Glass et al. 2002), which increases the complexity of classifying them, and the difficulties of excluding variations owing to random dispersion of personal judgments. The main strategy for reducing this threat was to involve several researchers in the data extraction. To enhance the quality of the data extraction form, it was jointly developed and refined by the three researchers and piloted twice to assess and augment the consistency of the extracted data as described in Section 3.4. Since many papers lacked sufficient information for unambiguous classification of them, it would have been preferable to perform parallel data extraction and cross-checking of the results between the researchers for all of the 38 papers on which data extraction was applied. However, a lack of resources made this impossible. Therefore, the researcher who extracted data for the remaining papers was to denote a short rationale explaining why the paper should be in a certain category. This was then used when the involved researchers debriefed and discussed uncertain classifications.

The possibility to generalize the findings in relation to the case used here are limited as the case study only involves two companies in the automotive domain. Using the results of similar case studies at other automotive companies may result in different findings, as they most likely face different needs. However, between 1999 and 2010, VCC played a leading role in the development of software-intensive systems at Ford Motor Company, which is one of the world's major car manufacturers, and VTC is one of the world's largest producers of heavy vehicles. Furthermore, both companies are organized as matrix organizations and their development of software-intensive systems is more or less guided by the V-model (ABG 1997) process that follows the overall vehicle development system, with its milestones (gates) for decision-making in a vehicle program (see also: Pernstal et al. 2012). This industrial setting is commonly used among automakers (Broy et al. 2007; Charfi and Selami 2004). Thus, we believe that our characterization and evaluation of the state of the art in these industrial settings are relevant for several companies, at least automotive companies.

The reliability threats are also a major concern in SMSs. To obtain reliability in this study, three researchers were involved in the investigation. They continuously documented and updated the research procedures in a review protocol (e.g., inclusion or exclusion procedures and piloting) as well as specifications (e.g., inclusion or exclusion criteria and data extraction forms) during the

study. The research methodology is presented and explained in Section 3 and the data extraction form is shown in Appendix B.
To summarize, it is possible we may have missed some relevant papers since the area of interest is multidisciplinary, where different terminologies and data sources are used. However, we have included as many studies as possible, and although we believe that the findings of this investigation can be slightly different in similar studies, it is unlikely that the overall findings will be different.

5 Conclusions

This paper presents an SMS that explores and examines the state of the art based on lean principles and practices, as used in large-scale development of software-intensive systems. The scope of this SMS was primarily justified by an expressed need from two case companies to support them in decisions on an SPI project—focusing on the inter-departmental interaction between PD and Man. In addition, the case companies, but also in many other companies in which software development is becoming an increasingly substantial part of PD, are introducing LPD, and they develop their products in multidisciplinary large-scale settings where communication and coordination within and across departments is critical. Thus, the focus of the analysis of the results was on the relationships between the reported state of the art and the broadly well-established lean principles inherent to LPD. Each principle was analyzed by using extracted data from the selected studies and data based on previous observations from the assessment of the case companies. This was a way to gauge the potential value of the results for the case companies, but also to identify the gaps in research on applying lean approaches in large-scale software-intensive systems development. Furthermore, to provide researchers with an overview of the status of the area and any research gaps, the selected studies were classified into a number of representative facets (e.g., research type and topic in SE) and visually summarized. To assist practitioners when seeking to adopt new "best" lean practices, and give researchers information about the quality of the studies reported, the degree of relevance and rigor for each study was also assessed and gauged. In total, data were extracted from 38 studies and the major findings were:

- Only 16 of the 38 studies (42 percent) clearly dealt with large-scale software development (RQ1). 11 out of these 16 studies reported on lean and five on hybrids building on combinations of lean and agile principles. Since it is recommended in research to combine traditional and agile methods in large-scale software development, it was unexpected that we could not find publications reporting on combinations of lean and plan-driven development (e.g., waterfall, RUP, and V-model). A majority of all the studies (24 out of 38) reported on the use of lean in general terms without specifying the particular practice adopted. Value stream mapping (two studies) and visual management (one study) were the only specific lean practices reported in the studies on large-scale software development.
- Classifying the studies into research types (RQ2.1) revealed a strong need for performing more studies on software development in real industrial cases where data for adopting lean principles and practices are systematically collected, analyzed, and evaluated. Most of the studies could be classified as experience papers (45 percent) and 10 of these studies dealt with large-scale software projects. Only 24 percent of all the studies could be classified as empirical and among these only four studies addressed large-scale development.
- The quality assessment of the studies (RQ2.2 and RQ2.3) showed that it is difficult for practitioners to judge the feasibility of the state of the art reported, and the possibilities to

replicate the studies and assess the validity of their results are limited. A majority (65 percent) of the studies report on work carried out in industries yielding a relatively high relevance. Of these, 14 studies addressed large-scale software development. However, about 75 percent of all the studies (28 out of 38) and nine studies on large-scale software development have relatively low values of rigor because of inadequate or missing descriptions of the study design and context, and evaluations of the validity.

- Classifying the studies into SE topics (RQ2.4) showed that there is little practical guidance for resolving problems identified in a specific KA (e.g., RE, testing, and coding), indicating that the reviewed area of interest is immature. Most of the studies primarily provide generic state of the art, since more than half of the studies could be classified into the broad KAs of engineering management (31 percent) and engineering process (24 percent). An almost identical pattern could also be seen for the studies on large scale development and none of these studies is classified into the KAs of configuration management, maintenance, design, and quality.
- The analysis of the relationships between LPD principles and the reported state of the art (RQ3) revealed that most of the studies focused on reducing waste and creating flow in the PD process (Principle 3). This was followed by visual management (Principle 12) and defining customer value (Principle 1). Furthermore, none of the studies on large-scale development had a strong relationship to the role of product manager (Principle 5), supplier integration (Principle 8), building a culture for continuous improvement (Principle 10), or tools for standardization and organizational learning (Principle 13).
- The deeper analysis showed that there is an overall lack of research that investigates the exploitation of LPD principles in the context of large-scale software development. The main contributions of the studies focused on identifying and eliminating waste through, for example, uncovering and measuring inventories of software development artifacts (e.g., requirements), and VSM. Establishing and maintaining flow in the software development process was also addressed by some studies, where incremental and concurrent development was most frequently suggested.
- When performing literature reviews on research across scientific fields (here SE, systems and manufacturing engineering, and management), the results indicate that it is difficult to obtain desired precision while at the same time ensuring the coverage of the search. The search in the selected databases resulted in a total amount of 10,230 papers, which were left for subsequent steps in the inclusion process. Of these, only 38 papers were selected for the subsequent data extraction. The main reasons are that different terminology is commonly used for the same notion, or terms are too general without the ability to filter out irrelevant studies.

We conclude that the current state of the art in research, offering specific advice to industry professionals pursuing improvements in large-scale software development, by applying lean principles and practices, is scarce. Furthermore, the implication for future research is that there is a strong need for more rigorous studies on the benefits of LPD. This includes a need to explicitly map what must be added to LPD and how the base principles must be changed and extended in order to support lean-oriented industrial sectors where the share of software in the products is rapidly growing.

Appendix A. List of included studies

[S1] A.L. Alwardt, N. Mikeska, R.J. Pandorf, P.R. Tarpley, A lean approach to designing for software testability. Proceedings of IEEE AUTOTESTCON, pp. 178-183, 2009.
[S2] S.W. Ambler, Agile software development at scale, Lecture Notes in Computer Science vol. 5082, Springer, 2008, pp. 1-12.
[S3] M. Aoyama, Managing the concurrent development of large-scale software systems, International Journal of Technology Management 14(1997) 739-765.
[S4] R. Benefield; Agile Deployment: Lean Service Management and Deployment Strategies for the SaaS Enterprise, Proceeding IEEE 42nd Hawaii International Conference on System Sciences, HICSS '09, pp.1-5, 2009.
[S5] M. Dall'Agnol, A. Janes, G. Succi, E. Zaninotto, Lean management—A metaphor for extreme programming?, Lecture Notes in Computer Science, vol. 2675, Springer, 2003, pp. 26-32.
[S6] E. Danovaro, A. Janes, G. Succi, Jidoka in software development, Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA, pp. 827-830, 2008
[S7] V. Gruhn, C. Schäfer, No-frills software engineering for business information systems experience Report, Proceedings of the eigth conference on New Trends in Software Methodologies, Tools and Techniques , SoMe_T 09, pp. 93-105, 2009
[S8] J. Heidenberg, I. Porres, Metrics functions for kanban guards, Proceedings 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS, pp. 306-310, 2010.
[S9] M. Ikonen, Leadership in Kanban software development projects: A quasi-controlled experiment, Lecture Notes in Business Information Processing, vol. 65, Springer, 2010, pp.85-98.
[S10] M. Ikonen, P. Kettunen, N. Oza, P. Abrahamsson, (2010). Exploring the sources of waste in Kanban software development projects. Proceedings of the 36th IEEE EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA, pp. 376-381, 2010.
[S11] B. Ippolito, E. Murman, Improving the software upgrade value stream, Proceeding of the 43rd AIAA Aerospace Sciences Meeting and Exhibit—Meeting Papers, pp. 4791-4803, 2005.
[S12] A. Janes, G. Succi, To pull or not to pull. Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA, pp. 889-894, 2009.
[S13] P. Kettunen, (2010), A tentative framework for lean software enterprise research and development. Lecture Notes in Business Information Processing, vol. 65, Springer, 2010, pp. 60-71.
[S14] F. Kinoshita, Practices of an agile team. Proceedings of the Agile Conference 2008, AGILE'08, IEEE, pp. 373-377, 2008.
[S15] F.J. Lacoste, Killing the gatekeeper: introducing a continuous integration system. Proceedings of the Agile Conference 2009, AGILE'09, IEEE, pp. 387-392, 2009.
[S16] P. Linecar, D. Preston, Kaizen: a review of Japanese approaches to IT, Proceedings First InternationalConference on Software Quality Management, pp. 903-909, 1993.
[S17] V. Mandic, M. Oivo, P. Rodríguez, P. Kuvaja, H. Kaikkonen, B. Turhan, What is flowing in lean software development?, Lecture Notes in Business Information Processing, vol. 65, Springer, 2010, pp. 72-84.
[S18] M. Mehta, D. Anderson, D. Raffo, Providing value to customers in software development through lean principles, Software Process Improvement and Practice, 13 (2008) 101-109.
[S19] P. Middleton, Lean software development: two case studies, Software Quality Journal 9 (2001) 241-252.
[S20] P. Middleton, P.S. Taylor, A. Flaxel, A. Cookson, Lean principles and techniques for improving the quality and productivity of software development projects: A case study, International Journal of Productivity and Quality Management 2 (2007) 387-403.
[S21] R. Morien, Agile management and the Toyota way for software project management, Proceedings of the 3rd IEEE International Conference on Industrial Informatics, Perth, Australia, pp. 516-522, 2005.
[S22] S. Mujtaba, R. Feldt, K. Petersen, Waste and lead time reduction in a software product customization process with value stream maps, Proceedings of the Australian Software Engineering Conference, ASWEC, pp. 139-148, 2010.
[S23] E. Parnell-Klabo, Introducing lean principles with Agile practices at a fortune 500 company, Proceedings of the Agile Conference 2006, IEEE, pp. 232-239, 2006.
[S24] G.I.U.S, Perera, M.S.D Fernando, Enhanced agile software development—Hybrid paradigm with LEAN practice, Proceedings of the 2nd International Conference on Industrial and Information Systems, ICIIS 2007, pp 239-243, 2007.

[S25] K. Petersen, C. Wohlin, Software process improvement through the Lean Measurement (SPILEAM) method, Journal of Systems and Software 83 (2010) 1275-1287.
[S26] D. Raffo, D.J. Anderson, R. Harmon, Integrating lean principles with value based software engineering, Proceedings of Technology Management for Global Economic Growth 2010, PICMET'10, pp. 1-10, 2010.
[S27] H. Rudolf, F. Paulisch, (2010). Experience report: Product creation through lean approaches, Lecture Notes in Business Information Processing, vol. 65, Springer, 2010, pp. 104-110.
[S28] T. Sekimura, T. Maruyama, Development of enterprise business application software by introducing Toyota production system, Fujitsu Scientific and Technical Journal 42 (2006) 407-413.
[S29] C.M. Shinkle, C.M. (2009). Applying the Dreyfus model of skill acquisition to the adoption of kanban systems at Software Engineering Professionals (SEP), Proceedings of the Agile Conference 2009, AGILE'09, IEEE, pp. 186-191, 2009.
[S30] H. Smits, The impact of scaling on planning activities in an agile software development context, Proceedings of the 40th Annual Hawaii International Conference on System Sciences, HICSS 2007, 2007.
[S31] M. Staron, W. Meding, B. Söderqvist, A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation, Information and Software Technology 52 (2010) 1069-1079.
[S32] J.M. Sutton, Lean software for the lean aircraft, Proceedings of theAIAA/IEEE Digital Avionics Systems, pp. 49-54, 1996.
[S33] M. Taipale (2010). Huitale—A story of a Finnish lean startup, Lecture Notes in Business Information Processing, vol. 65, Springer, 2010, pp. 111-114.
[S34] M. Thias, B. Cohen, The power of proximity: how colocated lean-agile teams deliver better project outcomes with fewer resources, Cutter IT Journal, 22 (2009) 37-43.
[S35] A.C. Tonini, F.J.B. Laurindo, M.M. De Spinola, An application of six sigma with lean production practices for identifying common causes of software process variability, Proceedings of the International Conference on Management of Engineering and Technology, IEEE, pp. 2482-2490, 2007.
[S36] K. Vilkki, K. (2010). When agile is not enough, Lecture Notes in Business Information Processing, vol. 65, Springer, 2010, pp 44-47.
[S37] B. Vodde, Measuring continuous integration capability, CrossTalk 21 (2008) 22-25.
[S38] E.R. Willeke, The Inkubook experience: A tale of five processes, Proceedings of the Agile Conference 2009, AGILE'09, IEEE, pp. 156-161, 2009.

Appendix B. Data extraction form

Section A—General data

ID	Attribute
A1	Title
A2	Author(s)
A3	Year
A4	Source
A5	Study purpose/objectives
A6	Domain (e.g., Automotive, Telecom, Web, etc.)
A7	Success/Failure?
A8	Does the paper repeat a work that has already been reviewed in another paper?
A9	Focus on lean or inter-departmental interaction?
A10	Focused method/practice
A11	Clearly and explicitly dealing with large scale software development?

Section B—Type of Research (Wieringa et al. 2006). What type of research is conducted?

ID	Type	Description
B1	Validation Research	Techniques investigated are novel and have not yet been implemented in practice.Techniques used are for example experiments, i.e., work done in the lab.
B2	Evaluation Research	Techniques are implemented in practice and an evaluation of the technique is conducted. That means, it is shown how the technique is implemented in practice (solution implementation) and what are the consequences of the implementation in terms of benefits and drawbacks (implementation evaluation). This also includes to identify problems in industry. Evaluation is formal i.e. old treatment is compared to new treatment (e.g. Lean practices). There must be some sort collection of empirical data (e.g. interviews, questionnaires or measurement of dependent variables) that are compared (e.g., defects throughput, leadtime etc.)
B3	Solution Proposal	A solution for a problem is proposed, the solution can be either novel or a significant extension of an existing technique. The potential benefits and the applicability of the solution is shown by a small example or a good line of argumentation.
B4	Philosophical Papers	These papers sketch a new way of looking at existing things by structuring the field in form of a taxonomy or conceptual framework.
B5	Opinion Papers	These papers express the personal opinion of somebody whether a certain technique is good or bad, or how things should been done. They do not rely on related work and research methodologies.
B6	Experience Papers	Experience papers explain on what and how something has been done in practice. It has to be the personal experience of the author. Experience differs from evaluation as it is no explicit comparison between the treatments, for example, you implement lean but you really dont no what the old process was.

Section C–Relevance of State-Of-The-Art (Ivarsson and Gorschek 2010). What is the relevance of the study?

ID	Aspect	Contribute to relevance (1)	Do not contribute to relevance (0)
C1	Subject	The subjects used in the study are representative of the intended users of the state-of-the-art.	The subjects used in the study are not representative of the intended users of the state-of-the-art (e.g., students, researchers or subjects not mentioned.
C2	Context	The study is performed in a setting that is representative of the intended setting.	The study is not performed in a setting that is representative of the intended setting (e.g., laboratory).
C3	Scale	The scale of the state-of-the-art used in the study is of realistic scale, i.e. industrial scale.	The scale of the state-of-the-art used in the study is of unrealistic scale (e.g., down-scaled industrial, toy example).
C4	Research Method	The research method used is designated for investigations of real world situation (e.g., action research, case studies, surveys).	The research method used is not designated for investigations of real world situation (e.g., conceptual analysis, laboratory experiment).

Section D—Rigor of State-Of-The-Art (Ivarsson and Gorschek 2010). What is the rigor of the study?

ID	Aspect	Strong description (1)	Medium description (0,5)	Weak description (0)
D1	Context	The context is described to the degree where a reader can understand and compare it to another context. This involves descriptions of context facets and their related elements, e.g., product (maturity, size), process (activities, workflow), people (roles, competencies).	The context in which the study is performed is briefly described to the degree to which a reader can understand and compare it to another context.	There is no description of the context in which the study was performed
D2	Study design	The study design is described to the degree to which a reader can understand the design with regard to, for example, treatment, variables, sampling etc.	The study is briefly described e.g. we implemented tool x in department c for n months and then department b etc.	There is no description of the study design.
D3	Validity	The validity threats are described and discussed in detail (e.g. external, internal, construct and construct validity, and reliability) and measures to limit them are presented.	The validity of the study is mentioned but not detailed.	There is no discussion of the threats to the study's validity

Section E–SWEBOK Knowledge Areas (KA) (http://www.computer.org/portal/web/swebok/html/ch1). What KA(s) can the contributions of the paper explicitly and clearly be mapped to?

ID	KA	Description
E1	Requirements	The Software Requirements (KA) is concerned with the elicitation, analysis, specification, and validation of software requirements.
E2	Design	Software design is defined in [IEEE610.12-90] as both "the process of defining the architecture, components, interfaces, and other characteristics of a system or component" and "the result of [that] process". Viewed as a process, software design is the software engineering life cycle activity in which software requirements are analyzed in order to produce a description of the software's internal structure that will serve as the basis for its construction. More precisely, a software design (the result) must describe the software architecture - that is, how software is decomposed and organized into components - and the interfaces between those components. It must also describe the components at a level of detail that enable their construction.
E3	Construction	The term software construction refers to the detailed creation of working, meaningful software through a combination of coding, verification, unit testing, integration testing, and debugging.
E4	Testing	Testing is an activity performed for evaluating product quality, and for improving it, by identifying defects and problems. Software testing consists of the dynamic verification.
E5	Maintenance	Software maintenance is defined as the totality of activities required to provide cost-effective support to software. Activities are performed during the pre-delivery stage, as well as during the post-delivery stage. Pre-delivery activities include planning for post-delivery operations, for maintainability, and for logistics determination for transition activities. Post-delivery activities include software modification, training, and operating or interfacing to a help desk.
E6	Configuration management	Configuration management (CM) is the discipline of identifying the configuration of a system, e.g. specific versions of hardware, firmware, or software items combined according to specific build and at distinct points in time for the purpose of systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the system life cycle.
E7	Engineering management	Software Engineering Management can be defined as the application of management activities—planning, coordinating, measuring, monitoring, controlling, and reporting—to ensure that the development and maintenance of software is systematic, disciplined, and quantified (IEEE610.12-90). Examples: organizational policies and standards provide the framework in which software engineering is undertaken, and the notion of project management involving project integration management, project scope management, project time management, project cost management, project quality management, project human resource management, and project
E8	Engineering process	The software engineering process is concerned with the definition, implementation, assessment, measurement, management, change, and improvement of the software life cycle processes themselves. Measurement should be in KA Engineering management unless it is measurement of the process cause then it goes into KA Engineering Process.
E9	Engineering tools and methods	This KA includes specific methods and tools whithin any other KA . When the focus is on the tool or method, rather than on the KA that the tool/method helps in/with, it belongs in this KA.
E10	Quality	Software quality in this KA cover static techniques, those which do not require the execution of the software being evaluated (e.g., inspection), while dynamic techniques are covered in the Software Testing KA.

Section F—Principles in Lean Product Development (LPD) (Morgan and Liker 2006). What relationships are there between the LPD-principle(s) and the contributions of the paper?

ID	Principle	Description	Strong relationship (++)	Weak relationship (+)	No relationship (0)
			There is a clear and explicit relationship between the contributions of the paper and the principle.	It is possible to deduce a relationship between the contributions of the paper and the principle but it is not clear and explicit.	There is no relationship between the conributions of the paper and the principle.
F1	Establish customer-defined value to separate value –added activity from waste.	To create a lean PD process, it is important to establish a customer defined value as a first step. Once these values have been identified, diffused and understood throughout the whole PD-organization it is possible eliminate waste.			
F2	Front-Load the Product Development Process while there is maximum design space to explore alternative thoroughly.	Front loading the product development process involves solving problems at root cause level in early project phases aiming to eliminate late engineering changes like "quick fixes" and patches that rarely result in increased product or process performance. This is achieved through concurent and multidisciplinary approaches such as Set Based Concurrent Engineering (SBCE).			
F3	Create a leveled Product Development Process flow	Leveling out the PD flow involves elimination of waste (everything that does not contribute to the value for the customer) in the product development flow. The total PD value stream is examined aiming to eliminate non value adding activities that occur between development steps (e.g. unnecessary handovers of documents and reinvention instead of standardization of components ("not invented here").			
F4	Utilize rigorous standardization to reduce variation, and create flexibility and predictable outcomes.	Standardization has a large influence on PD since it reduces variation which enables increased flexibility and predictable outcomes. There are three categories of standardization: 1) design standardization, 2) process standardization and 3) engineering skill-set standardization.			
F5	Develop a chief engineer (CE) system to integrate development from start to finish.	The CE is appointed by the top management immediately after a new program has been decided where the CE are considered as the owner of the product responsible for the whole development process from concepts to launch.			
F6	Organize to balance functional expertise and cross-functional integration.	Creating efficient PD organizations by combining the benefits of product and functional focused structures in a matrix organization allowing simultaneous attention to functional and program demands. This matrix structure contains the program based organizations in the lateral direction and the deep specialized functional departments in the vertical.			
F7	Develop towering technical competence in all engineers	The necessity to use a rigorous recruitment process, mentoring and on-the-job-training (OJT) in a structured way. For example, in order to have the capability to technically challenge PD engineers, the reqruitment and training of manufacturing engineers should be equally comprehensive.			
F8	Fully integrate suppliers into the Product Development System.	Comprises such as a high degree of supplier involvement which implies early involvement of suppliers in PD, rigorous selection of suppliers and that suppliers are committed to continuously maintain and develop their engineering and manufacturing capabilities to meet the demands of the ordering company (e.g., OEM).			
F9	Build in learning and continuous improvement.	To achieve continuous improvement (kaizen) of products/processes, it is important to recognize and encourage learning and understanding of technologies and processes where both tacit and explicit knowledge are developed, diffused and maintained in the organization. For example, a cognitive learning approach is emphasized where problems are viewed as opportunities and it is essential to bring the problems to the surface and solve them as early as possible.			
F10	Build a culture to support excellence and relentless improvement.	Culture embraces a fairly stable set of taken-for-granted assumptions, shared beliefs, meanings, and values in an organization that govern the members' operations and enables the organization to rely less on formal lean control systems. Encouraging a mindset among the employees based on customers come always first and there is always more to learn, understand and improve are examples of building a lean culture within an organization.			
F11	Adapt technology to fit your people and processes.	Tools and technology must be customized based on organizational needs. For example, seamless integration of new technologies facilitating incorporation with existing systems/tools and adaptation to established processes and not vice versa.			
F12	Align your organization through simple visual communication.	Deals with the organization's capability of efficiently coordinating complex communication (e.g., requirements, test results, project status reportsmanufacturing constraints etc..) between teams and across functions in PD.			
F13	Use powerful tools for standardization and organizational learning.	To build learning organizations, it is necessary to deploy tools that support development, diffusion and preservation of both explicit and tacit knowledge.			

Appendix C. Relationships between lean principles in LPDS (Morgan and Liker 2006) and selected studies

Table C.1

Sub-system	Principle	Description	Strong relationship*		Weak relationship	
			Studies	Freq.	Studies	Freq.
Process	1 .Establish customer-defined value to separate value-added activity from waste.	To create a lean PD process, it is important to establish a customer defined value as a first step. Once these values have been identified, diffused and understood throughout the whole organization, it is possible eliminate waste.	S7,S11,S13, S14S20,S21, S26,S27 S33	9(4)	S8, S22, S32	3(2)
	2. Front-load the PD process while there is maximum design space to explore alternative thoroughly.	Involves problem-solving at root cause level in early project phases. The aim is to eliminate late engineering changes like 'quick fixes' and patches that rarely result in increased product or process performance.	S7,S8,S12, S18,S26,S27 ,S33	7(1)	S4,S14 S17,S35	4(0)
	3. Create a leveled PD process flow.	Involves eliminating waste (everything that does not contribute to the value for the customer) and establishing flow (regular pace) in the PD process. The total PD value stream is examined with the aim to eliminate non-value adding activities that occur between development steps such as unnecessary handovers of documents and reinvention instead of standardization of components. Flow is created by incremental development where work is broken down into suitable tasks.	S3,S6,S7,S8, S9,S10,S11, S12,S13S14, S15,S17,S18 S19,S20,S22 ,S23,S25,S2 8,S29,S30,S 32,S34,S37, S38	25 (11)	S1,S5 S21,S27 S31,S33	6(3)
	4. Utilize rigorous standardization to reduce variation, and create flexibility and predictable outcomes.	Standardization has a large influence on PD since it reduces variation that enables increased flexibility and predictable outcomes. There are three categories of standardization: design standardization, process standardization, and engineering skill-set standardization.	S32	1(1)	S3,S18 S20,S28	4(3)
Skilled People	5. Develop a chief engineer system to integrate development from start to finish.	The top management appoints a chief engineer immediately after a new program has been decided upon. The chief engineer is considered to be the owner of the product and is responsible for the whole development process from concepts to launch.		0	S11	1(1)
	6. Organize to balance functional expertise and cross-functional integration.	Creating efficient PD organizations by combining the benefits of product and functional focused structures in a matrix organization. This allows simultaneous attention to functional and program demands.	S36	1(1)	S2,S27	2(2)
	7. Develop towering technical competence in all engineers.	The necessity to use a rigorous recruitment process, mentoring and on-the-job-training (OJT) in a structured way. For example, in order to have the capability to technically challenge design engineers, the recruitment and training of manufacturing engineers should be equally comprehensive.	S11,S16,S29 , S23	4(2)	S1,S17 S20	3(2)
	8. Fully integrate suppliers into the Product Development System.	Comprises a high degree of supplier involvement. This implies early involvement of suppliers in PD, a rigorous selection of suppliers, and that suppliers are committed to continuously maintain and develop their engineering and manufacturing capabilities in order to meet the demands of the ordering company, i.e. original engineering manufacturer (OEM).		0		0
	9. Build in learning and continuous improvement.	To achieve continuous improvement (Kaizen) of products or processes, it is important to recognize and encourage learning and understanding of technologies and processes where both tacit and explicit knowledge are developed, diffused and maintained in the organization.	S3,S13,S15, S16,S22,S26 ,S27,S35	8(4)	S17,S23 S25,S28 S29,S34	6(3)
	10. Build a culture to support excellence and relentless improvement.	The culture embraces a fairly stable set of assumptions that are taken-for-granted, shared beliefs, meanings, and values in an organization that govern the members' operations and enables the organization to rely less on formal lean control systems.	S16, S19	2(0)	S4,S18 S27	3(2)
Tools & Technology	11. Adapt technology to fit your people and processes.	Tools and technology must be customized based on organizational needs. This means that the integration of new technologies facilitating incorporation with existing systems or tools and adaptation to established processes should be seamless, and not vice versa.	S31	1(1)	S3,S7 S13,S19 S24,S27 S28,S38	8(4)
	12. Align your organization through simple visual communication.	Deals with the organization's capability of effectively coordinating complex communication such as requirements, test results, project status reports and manufacturing constraints between teams and across functions in the PD process.	S6,S8,S11, S14,S19,S20 S25,S28,S31 ,S38	10(5)	S7,S9 S10,S18 S27,S29 S35,S37	8(2)
	13. Use powerful tools for standardization and organizational learning.	To build learning organizations, it is necessary to deploy tools that support development, diffusion and preservation of both explicit and tacit knowledge, on which evolving standards are based.		0	S20,S25	2(2)

*Frequencies of relationships to studies on large-scale software development are placed in brackets.

References

ABG. (1997). V-Model: Development Standard for IT-Systems of the Federal Republic of Germany. Lifecycle Process Model.

Abrahamsson, P., Warsta, J., Siponen, M.T., Ronkainen, J. (2003). New directions on agile methods: a comparative analysis. In *The 25th International Conference on Software Engineering*, pp. 244-254.

Abran, A., Moore J.W. et al. (2004). Guide to the software engineering body of knowledge (SWEBOK®), IEEE Computer Society Guide.

Almefelt, L., Berglund, F., Nilsson, P., & Malmqvist, J. (2006). Requirement management in practice: findings from an empirical study in the auto-motive industry. *Research in engineering design, 17*, 3.

Andersen, B., & Fagerhaug, T. (2000). Roo1 Cause Anlysis: Simplified Tools and Techniques. Milwaukee, Wisconsin: ASQ Quality Press.

Aoyama, M. (1997). Agile Software Process Model. In *The 21st Annual International Computer Software and Applications Conference, COMPSAC '97*. Washington DC, USA.

Automotive SIG. (2010). The SPICE User Group, Automotive SPICE Process Assessment Model v2.5 and Process Reference Model v4.5, http://www.automotivespice.com.

Basili, V.R. (1985). *Quantitative Evaluation of Software Methodology*. University of Maryland. College Park, MD.

Beck, K. (2004). *Extreme Programming Explained: Embrace Chage* (2nd ed). Addison-Wesley.

Bjarnason, E., Wnuk, K., Regnell, B. (2009). Requirements are slipping through the gaps—A case study on causes & effects of communication gaps in large-scale software development. In *Proc. IEEE 19th International Requirements Engineering Conference*, pp. 37-460.

Brooks, F.P. (1987). No Silver Bullet Essence and Accidents of Software Engineering. *IEEE Computer* 20, 10-19.

Broy, M., Kruger, I.H., Pretschner, A., & Salzmann, C. (2007). Engineering Automotive Software. *Proceedings of the IEEE*, *95*, 2.

Boehm, B. (2002). Get ready for agile methods, with care. *IEEE Computer 35*, 64–69.

Carlsson, M. (1991). Aspects of the Integration of Technical Functions for efficient Product Development. *R&D Management, 21*, pp. 55-66.

Cawley, O., Wang, X., Richardsson, I.. (2010). Lean/Agile Software Development Methodologies In Regulated Environments - State of the art. *Lecture Notes in Business Information Processing*, *Vol. 65*, 31-36, Springer-Verlag, Berlin Heidelberg.

Charfi, F., Sellami, F. (2004). Overview on Dependable Embedded Systems in Modern Automotive. In *Proc.of the 2004 IEEE Int. Conference on Industrial Technology*.

Clark, K.B., Fujimoto, T. (1991). *Product Development Performance: Strategy, Organization and Management in the World Auto Industry*. Harvard Business School, Boston..

CMMI. (2010). Capability Maturity Model Integration Version 1.3. Technical Report CMU/SEI-2010-TR-033. Software Engineering Institute (SEI). http://www.sei.cmu.edu/cmmi/.

Conboy, K. (2009). Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research, 20*, 329–354.

Coplien, J., & Bjornwig, G. (2010). *Lean architecture for agile software development*. West Sussex, UK, John Wiley & Sons Ltd.

Curtis, B., Krasner, H., Iscoe, N. (1988). A field study of the software de-sign process for large systems. *Communications of the ACM 31*.

Dannenberg, J., & Kleinhans, C. (2004). The coming age of collaboration in the automotive industry. *Mercer Manage. J., 18*, 88–94.

de Almeida Biolchini, J.C., Mian, P.G., Natali, A.C.C., Conte, T.U., Travassos, G.H. (2007). Scientific research ontology to support systematic review in software engineering. *Advanced Engineering Informatics 21*, 133–151.

Dybå, T., Dingsøyr, T. (2008). Empirical Studies of Agile Software Development: a Systematic Review. *Journal of Information and Software Technology 50*, 833-859.

Dybå, T., T Dingsøyr, T., Hanssen, G.K. (2007). Applying Systematic Reviews to Diverse Study Types: An Experience Report. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 225-234.

Dybå, T., Kitchenham, B., Jorgensen M. (2005). Evidence-based software engineering for practitioners. *Software, IEEE 22*, 58-65.

Eppinger, S.D., Whitney, D.E., Smith, R.P., Gebala, D.A. (1994). A Model-Based Method for Organizing Tasks in Product Development. *Research in Engineering Design 6*, 1–13.

Fayad, M.E., Laitnen, M. (1997). Process assessment considered wasteful. *Commun. ACM 40*, 125-128.

Fleiss, J. (1971). Measuring nominal scale agreement among many raters. *Psychological Bulletin 76*, 378–382.

Fricker, S., Gorschek, T., Byman, C., Schmidle, A. (2010). Handshaking with Implementation Proposals: Negotiating Requirements Understanding. *IEEE Software 27*, 72-80.

Glass, R.L., Vessey, I., Ramesh, V. (2002). Research in software engineering: an analysis of the literature. *Information and Software Technology 44*, 491-506 .

Hibbs, C., Jewett, S., Sullivan, M. (2009). *The Art of Lean Software Development: A Practical and Incremental Approach*. O'Reilly Media, Inc.

Highsmith, J. (2002). *Agile Software Development Ecosystems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Hiranabe, K. (2008). *Kanban Applied to Software Development: from Agile to Lean*. Available from: http://www.infoq.com/articles/hiranabe-lean-agile-kanban (accessed on Dec. 12).

Hoffman, D., Weiss, D. (2001). *Software Fundamentals: Collected Papers by David L. Parnas*. Upper Saddle River, NJ: Addison Wesley.

Holmdahl, L. (2010). Lean Produktutveckling på Svenska, Göteborg, ISBN 978-91-979196-0-9.

Ivarsson, M., Gorschek, T. (2009). Technology Transfer Decision Support in Requirements Engineering Research: A Systematic Review of REj. *Requirements Engineering journal, 14*, 155-175.

Ivarsson, M., Gorschek, T. (2011). A Method for Evaluating Rigor and Industrial Relevance of Technology Evaluations. *Empirical Software Engineering, 16*, 365-395.

Kaner, C. (2003). Blog Archive » SWEBOK Problems, Part 2, 2003-06-27, http://kaner.com/?p=27. Accessed 7 Dec. 2012.

Kano, N. (1996). *Guide to TQM in Service Industries*. Asian Productivity Org.

Karlsson, C., Åhlström, P. (1996). Assessing changes towards lean production. *International Journal of Operations & Production Management 16*, 24-41.

Karlström, D., Runesson, P. (2005). Combining agile methods with stage-gate project management. *IEEE Software, 22*, 43–49.

Kennedy, M.N., Harmon, K., Minnock, E. (2008). *Ready, Set, Dominate –implementing Toyota's set based Learning for Developing Products ... and nobody can catch you*. Richmond, VA: Oaklea Press.

Kitchenham, B., Charters, S. (2007). *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. Technical Report EBSE 2007-01, School of Computer Science and Mathematics, Keele University.

Kraut, R., & Streeter L.A. (1995). Coordination in software development. *Communications of the ACM, 38*, 69-81.

Kuilboer, J.P., & Ashrafi, N. (2000). Software Process and Product Improvement: An Empirical Assessment. *Information and Software Technology, 42*, 27-34.

Lakemond, N., Johansson, G., Magnusson, T., Safsten K. (2007). Interfaces between technology development, product development and production: critical factors and a conceptual model. *International Journal of Technology Intelligence and Planning 3*, 317-330.

Landis, J.R., Koch G.G. (1977). The measurement of observer agreement for categorical data. *Biometrics 33*, 159–174.

Layman, L., Williams, L., Cunningham, L. (2004). Exploring extreme programming in context: an industrial case study. *Agile Development Conference 2004*.

Lean Software Development. (2012). *IEEE Software 95*.

Liker, J.K. (2004). *The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer*. McGraw-Hill, New York.

Mannaro, K., Melis, M., Marchesi, M. (2004). Empirical analysis on the satisfaction of IT employees comparing XP practices with other software development methodologies. In *Extreme Programming and Agile Processes in Software Engineering, Lecture Notes in Computer Science, vol. 309*2, Springer Verlag, 166–174.

Miller, E.J., & Rice, A.K. (1967). *Systems of organizations*. London Tavistock.

Morgan, J.M., & Liker, J.K. (2006). *The Toyota Product Development System: Integrating People, Process, and Technology*. Productivity Press, New York.

Murman, E. (2004). Lean Aerospace Initiative.Eng Syst Symp. MIT, Cambridge, MA. Available at http://esd.mit.edu/symposium/pdfs/day1-2/murman-slides.pdf, 16 slides.

Negroni, D.Y., Trabasso L.G. (2009). A quality improving method to assist the Integrated Product Development process. In *Proc. of the 17th International Conference on Engineering Design (ICED 09,)* Palo Alto, CA, pp 127-136.

Nihtilä, J. (1999). R&D–Production integration in the early phases of new product development projects. *Journal of Engineering and Technology Management, 16*, 55-81.

Ohno, T. (1988). *Toyota Production System. Beyond Large-Scale Production*. Productivity Press, Portland.

Pernstal, J., Magazinius, A., Gorschek, T. (2012). A Study Investigating Challenges in the Interface between Product Development and Manufacturing in the Development of Software Intensive Automotive Systems. *International Journal of Software Engineering and Knowledge Engineering 20,* 965-1004.

Petersen, K. (2010). Is Lean Agile and Agile Lean? A Comparison Between Two Software Development Paradigms. In Modern Software Engineering Concepts and Practices: Advanced Approaches; Ali Dogru and Veli Bicer (Eds.), IGI Global.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M. (2008). Systematic Mapping Studies in Software Engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering*, 71-80.

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: an agile toolkit*. Addison-Wesley, Boston.

Reinertsen, D.G. (2009). *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas Publishing, Redondo Beach, CA.

Schwaber, K., & Beedle, M. (2001*). Agile Software Development with Scrum*. Prentice Hall, Upper Saddle River.

Sobek, D.K., Ward, A.C., Liker, J.K. (1999). Toyota's Principles of Set-Based Concurrent Engineering. *Sloan Management Review 40*.

Sommerville, I. (2007). *Software Engineering*, Eighth ed. Addison-Wesley Publishers Ltd.

Svensson, H., Höst, M. (2005). Views from an organization on how agile development affects its collaboration with a software development team. *Lecture Notes in Computer Science, vol. 3547*, Springer Verlag, Berlin, 487–501.

Takeuchi, H., I. Nonaka, I. (1986). The new product development game. *Harvard Business Review*, January-February, 137-146.

Vandevelde, A., Van Dierdonk, R. (2003). Managing the design-manufacturing interface. *Int. J. of Operations & Production Management 23*, 1326-1348.

Venkatesh Prasad, K., Broy, M., & Krueger, I. (2010). Scanning Advances in Aerospace & Automobile Software Technology. Proc. of the IEEE, 98, 510-514.

Wang, X., Conboy, K., Cawley, O. (2012). “Leagile” software development: An experience report analysis of the application of lean approaches in agile software development. *J. of Systems and Software 85*, 1287– 1299.

Ward, A.C. (2007). *Lean product and process development*. Lean Enterprise Institute, Cambridge, Mass. ISBN 978-1-934109-13-7.

Weber, M., Weisbrod, J. (2003). Requirements engineering in automotive development: experiences and challenges. *IEEE Software 20*, 16–24.

Wellington, C.A., Briggs, T., Girard, C.D. (2005). Comparison of student experiences with plan-driven and agile methodologies. In *Proceeedings of the 35th ASEE/ IEEE Frontiers in Education Conference 2005*.

Wheelwright, S.C., & Clark, K.B. (1994). Accelerating the design-build-test cycle for effective product development. *International Marketing Review 11*,32-46.

Wieringa, R., N.A.M. Maiden, N.A.M., Mead, N.R., Rolland C. (2006). Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering 11*, 102–107.

Womack, J., & Jones, D. (2003). *Lean Thinking: Banish Waste and Create Wealth in your Corporation*, Revised and updated edition. Simon & Schuster Ltd., London, UK.

Womack, J.P., Jones, D.T., Roos, D. (1990). *The Machine that Changed the World*. Rawson Associates, New York.

Zahran, S. (1998). *Software Process Improvement: Practical Guidelines for Business Success*. Reading, Addison-Wes.