Kupiainen, E., Mäntylä M. V., Itkonen J., "**Using Metrics in Agile and Lean Software Development - A Systematic Literature Review of Industrial Studies**", *Information and Software Technology,* 2015

Using Metrics in Agile and Lean Software Development – A Systematic Literature Review of Industrial Studies

Eetu Kupiainen[a], Mika V. Mäntylä[a,b,*], Juha Itkonen[a]

[a] *Department of Computer Science and Engineering, Aalto University, Finland*
[b] *Department of Information Processing Science, University of Oulu, Finland*

Abstract

Context: Software industry has widely adopted Agile software development methods. Agile literature proposes a few key metrics but little is known of the actual metrics use in Agile teams.
Objective: The objective of this paper is to increase knowledge of the reasons for and effects of using metrics in industrial Agile development. We focus on the metrics that Agile teams use, rather than the ones used from outside by software engineering researchers. In addition, we analyse the influence of the used metrics.
Method: This paper presents a systematic literature review (SLR) on using metrics in industrial Agile software development. We identified 774 papers, which we reduced to 30 primary studies through our paper selection process.
Results: The results indicate that the reasons for and the effects of using metrics are focused on the following areas: sprint planning, progress tracking, software quality measurement, fixing software process problems, and motivating people. Additionally, we show that although Agile teams use many metrics suggested in the Agile literature, they also use many custom metrics. Finally, the most influential metrics in the primary studies are Velocity and Effort estimate.
Conclusion: The use of metrics in Agile software development is similar to Traditional software development. Projects and sprints need to be planned and tracked. Quality needs to be measured. Problems in the process need to be identified and fixed. Future work should focus on metrics that had high importance but low prevalence in our study, as they can offer the largest impact to the software industry.

Keywords: Agile, Lean, metrics, measurement, systematic literature review, software engineering

*Corresponding author
Email addresses: eetu.kupiainen@gmail.com (Eetu Kupiainen), mika.mantyla@oulu.fi (Mika V. Mäntylä), juha.itkonen@aalto.fi (Juha Itkonen)

1. Introduction

Agile software development methods, such as Scrum [32], Extreme Programming [2], Lean Software Development [27], and Kanban [1], are already the most common software development approaches in industry [31]. In Agile methods, the focus is in lightweight working practices, constant deliveries, and customer collaboration over long planning periods, heavy documentation, and inflexible development phases [3].

Software metrics have been studied for decades and several literature reviews have been published. For example, Kitchenham [19] performs a mapping of the most cited software metrics papers and summarises recent review papers. However, according to the authors' best knowledge, there are no systematic literature reviews on the reasons for and effects of using metrics in the Agile software development context.

Agile methods follow certain pre-given metrics and a set of rigorous practices. Examples of Agile metrics are burn-down-charts, test-pass rates, and sustainable pace. These metrics, however, seem to differ from the Traditional measurement programs. For example, a team's velocity is used in XP to assign a sustainable amount of work to the team and plan the iteration contents accordingly, whereas traditional planning would set the team a productivity goal as given and track team performance against that goal. In Agile mindset, estimating is applied as a way to predict how much the team can get done to guide sprint planning—not as a target that should be achieved as closely as possible. Agile emphasises measuring progress in terms of working software over measuring intermediate work products (documents) and strives for making the measurement simple and immediate. Overall, Agile creates two types of conflicts to Traditional measurement approaches. First, the traditional approach of tracking progress against a pre-made plan and measurable goals [36] conflicts with the Agile value of embracing the change. Second, the standard quality measurement approaches, such as [15], propose a rather comprehensive set of metrics, which does not align well with the Agile principle of simplicity.

To exemplify this contrast between Agile and Traditional, we list the main differences in measurement in Traditional and Agile contexts:

- Traditional: controlling, outsider viewpoint, tracking deliverables, setting measurable goals, following a plan, large programs [36, 10]
- Agile: team in control, customer focus, simplicity, following a trend, fast feedback, responding to change [1, 33, 3]

While Agile and Lean development organizations seem to have well reasoned needs for metrics, the overall picture is not clear on what metrics Agile teams are using in practice, for what purpose, and with what effects. Empirical metric research in the context of Agile methods remains scarce, and in this study we aim at laying out the current state of practice of software metrics in industrial Agile software development based on the empirical literature. We make a systematic literature review (SLR) to investigate what metrics are used in

industrial projects. We further analyse the reported motivations, i.e., benefits of applying the metrics, as well as the effects of using metrics in Agile teams. In addition, the importance of the metrics is analysed to create understanding of the characteristics of high influence metrics in Agile and Lean contexts. We study Agile and Lean together, as comparison between them indicates that Agile and Lean share the same goals and rely on similar principles [25]. Therefore, in this study we view Lean as a special case of Agile software development, although some individuals would claim that it is the other way around.

Based on these goals we state the the following research questions:

- **Research Question 1:** *What metrics are used in industrial Lean and Agile software development?*
- **Research Question 2:** *What are the reasons for and effects of using metrics in industrial Lean and Agile software development?*
- **Research Question 3:** *What metrics have high influence in industrial Lean and Agile software development?*

In this SLR study we focus on research and experience reports that report empirical findings on real uses of metrics in Agile context. We only select papers in which the context is properly described to understand if a development team or software company is using the described metrics. We exclude cases where metrics are used purely for research or comparison purposes, i.e., metrics that are not used to support the software development work. We also require that the motivation, effects, or importance of the metrics is somehow addressed in the selected primary studies.

Previously, we have published the initial results of our SLR as a 7-page workshop paper [22]. The previous paper only presented the initial results of RQ2. In this work, the analysis of RQ2 has progressed resulting in better and more detailed results. Additionally, this work presents two new research questions RQ1 and RQ3. Finally, the discussion that helps the reader to interpret the results has been greatly extended.

This article is structured as follows. Section 2 provides background information about related concepts regarding this study. Section 3 describes how the systematic literature review was conducted. Section 4 reports the results of the study. Section 5 discusses the findings and how they map to prior research. Section 6 concludes the study.

2. Background and Related Work

In this section, we describe the background related to Agile software development and software measurement research, and introduce the key concepts used in this study. First, we introduce the Agile software development approach and its relationship to Lean software development. Then, we review the benefits of software measurement in general and give a brief overview of existing review studies on software metrics.

2.1. Agile software development

Agile software development has emerged to provide an alternative to plan-driven and often heavyweight methods. Agile methods share common values and principles [3]. Agile methods value individuals and interactions, working software, customer collaboration, and responding to change over processes, documentation, contracts, and plans. Agile development emphasises short development cycles, frequent deliveries, continuous face-to-face communication, and learning. Popular Agile development methods include Scrum [32] and Extreme Programming [2]. Lean Software Development (LeanSD)[27] and Kanban [1] share the similar values and principles with Agile methods.

Scrum [32] method is characterised by daily team meetings and development sprints. On a high level, the development is constructed from multiple subsequent sprints, where an increment of the software is developed. Sprints are planned by selecting items from a backlog and estimating the effort needed to complete each item selected for the sprint. During sprints, the team groups up every day for a daily scrum meeting, where the status of the tasks is tracked. At the end of the sprint, a sprint review and demo is organized. Learning is emphasized in every sprint with a sprint retrospective meeting.

Extreme Programming (XP) [2] emphasizes a set of principles and practices to the extreme. For example, automated unit testing, pair programming, and continuously refactoring the code base are made very rigorous practices to enable agile ways of working. Changes in business requirements can then be flexibly developed. Communication is efficiently handled with collocated teams, unit tests, pair-programming, and having a customer continuously available to provide information on business requirements.

LeanSD and Kanban can be seen as approaches where traditional Lean manufacturing [38] philosophies, principles, and tools are applied to software development. As such, it is not easy to separate Agile and Lean methods. A comparison of Agile and Lean principles, for example, has revealed them to be very similar [25].

> "Lean development further expands the theoretical foundations of Agile software development by applying well-known and accepted Lean principles to software development. But it goes further by providing thinking tools to help translate Lean principles into agile practices that are appropriate for individual domains." [27]

The modified lean principles used in LeanSD are:

1. *Eliminate Waste*
2. *Amplify Learning*
3. *Decide as Late as Possible*
4. *Deliver as Fast as Possible*
5. *Empower the Team*
6. *Build Integrity In*
7. *See the Whole*

Kanban is neither a software development lifecycle methodology nor a project management approach. Instead, Kanban can be applied to incrementally change and improve some underlying, existing process [1]. It is an evolutionary process model that allows each Kanban implementation to be different, suited for each context. On the other hand, LeanSD describes Kanban as part of one of its tools: "Pull Systems"[27]. Kanban, too, defines certain principles; for example, Kanban systems are always pull systems. Work is pulled to development only when there is capacity, compared with some other systems where work is pushed to development.

In this study, we scope our focus on Agile software development, but as the differentiation between Agile, Lean, and Kanban in software development methodologies is not clear or even meaningless in our case, we include in this research all cases where the applied method is described as being either Agile, LeanSD, Kanban, or some combination of these.

2.2. Software measurement

Metrics use has been motivated in the prior work by several authors giving possible reasons and effects of using metrics, in the traditional software engineering context, that might also be applicable to Agile software development: "*If you cannot measure it, you cannot improve it...*" is perhaps the most popular metrics motivation, originally given by Lord Kelvin, a mathematical physicist and engineer. According to Fenton and Pfleeger [10], we use metrics every day to understand, control and improve what we do and how we do it. Furthermore, Jones [17] states, based on the knowledge base of thousands of software projects, that the top-performing software companies, such as IBM and Microsoft, extensively use metrics in their business, while the lower-performing teams do not. Buse and Zimmermann [6] surveyed the information needs of software managers and developers at Microsoft and found that the most used information is related to quality, e.g., automated failure and bug reports, which are used by roughly 75% of the respondents. Common needs for metrics are related to supporting communication and decision making. Summarizing Pulford et al. [28] and Grady [12] gives us the following motivations for metrics use:

- Project planning and estimation
- Project management and tracking
- Understanding quality and business objectives
- Improved software development communication, processes, and tools

In addition to the software engineering literature motivating the use of metrics in software development in general, there are existing review articles on software metrics studies. There are a few mapping studies on software metrics, and Kitchenham [19] reports that there is a large body of research related to software metrics. Kitchenham [19] summarises four recent survey studies on software metrics. For example, two papers review fault prediction metrics

[7, 30], Purao and Vaishnavi [29] reviewed object-oriented product metrics, and Bellini et al. [4] presented a systematic review focusing on how the concepts and research in software measurement have been developed, as well as the implications of the research trends for research and practice.

There are many benefits to software metrics, such as being able to predict and improve many aspects of software projects and, in general, make better decisions. However, it is important to study the metrics, their use, and their benefits in context, because failure to understand the context will limit the understanding and the usefulness of the results in varying industrial contexts [19]. Also Radjenović et al. [30] concludes that more studies are needed in large industrial contexts to find metrics that are relevant to the industry and to understand which metrics should be used in a given context. Yet, none of the reviews of metrics literature of the previous paragraph focused on Agile methods or even classified the existing studies based on the software development approach.

The goal of this study is to research the metrics that are applied in the Lean and Agile software development contexts, based on empirical research. We also aim at understanding the reasons and effects of metric use, and at characterising the important metrics in these contexts.

3. Research Method

A systematic literature review (SLR) was chosen as a research method because the study is more about trying to understand a problem than trying to find a solution to it. Also, there was already existing literature that could be synthesised. An SLR is a research method originating from the field of medicine [18]. There are three main reasons for conducting an SLR [21]. First, to aggregate and synthesise existing knowledge regarding a research topic. Second, to identify gaps in earlier research. Third, to provide background information to start investigating a new research topic. Moreover, an SLR provides a repeatable research method which, when applied properly, should provide sufficient detail to be replicated by other researchers. Furthermore, the detailed documentation of the performed steps within the SLR enables in-depth evaluation of the conducted study.

An SLR is a trustworthy research method for multiple purposes. In this study, an SLR is used to perform a comprehensive study of empirical research on using metrics in Agile software development. Both quantitative and qualitative analysis methods are used to understand which metrics are used, why, what are the effects of metric use, and to characterise the importance of metrics in Agile context.

The guidelines provided by Kitchenham [18] were used as a basis to develop the SLR protocol. Additionally, [9] and [20] were used to further understand the challenges and opportunities of SLRs. The protocol was developed iteratively, performing first a small pilot study and iterating the details of the protocol in weekly meetings among the researchers. In addition, the validity of both

Table 1: Paper selection funnel

Stage	Amount of papers
Stage 1	774
Stage 2	163
Stage 3	30

the study selection and data extraction procedures was evaluated as described below.

In the following subsections, we describe the search including the primary study selection process, pilot study, data extraction procedures, data analysis, and data synthesis.

3.1. Search and selection process

The strategy for finding primary studies was the following:

- Stage 1: Automated search
- Stage 2: Selection based on title and abstract
- Stage 3: Selection based on full text. Data extraction and quality assessment.

Table 1 shows the selection funnel in terms of the number of papers after each stage. Scopus database [1] was used to find the primary studies with automated search. Keywords include popular Agile development methods and synonyms for the word "metric". The search was improved incrementally in three phases because some key papers and XP conferences were not found initially. The search strings, hits, and dates can be found in Appendix A.

The selection of the primary studies was based on the following inclusion criteria: *papers that present empirical findings on the industrial use and experiences of metrics in an Agile context.* Papers were excluded based on multiple criteria, mainly due to not conforming to requirements regarding empirical findings, Agility, and industrial context. The full criteria are listed in Appendix B.

In stage 1, Scopus was used as the only search engine, as it contained the most relevant databases IEEE and ACM. Also, it was able to find Agile and XP conference papers. Only the XP Conference 2013 was searched manually because it could not be found through Scopus.

In stage 2, papers were included and excluded based on their title and abstract. As the quality of abstracts can be poor in computer science [18], full texts were also skimmed through in case of unclear abstracts. Unclear cases

[1] http://www.scopus.com

were discussed among the researchers in weekly meetings, and an exclusion rule was documented if necessary.

The validity of the selection process was analysed by performing the selection for a random sample of 26 papers also by the second author. The level of agreement was "substantial" with Kappa 0.67 [23].

Stage 3 included multiple activities in one workflow. This included the selection by full text, data coding, and quality assessment. Once again, if there were unclear papers, they were discussed in the meetings. Also, the selection of 7 papers was conducted by the second author with an "almost perfect" agreement, Kappa 1.0 [23].

3.2. Pilot study

We conducted a pilot study after the first database searches to refine the aim of the research and get familiar with the research method. Moreover, it was possible to modify the method and tools before applying them to the full set of primary studies.

Fifteen papers were selected for the pilot; 5 by relevance, 5 by number of citations, and 5 by random selection. Based on the pilot study, a few improvements to the SLR protocol were made. First, the selection by title and selection by abstract steps were joined together to improve the reliability of the first selection round. Second, the quality assessment checklist was decided based on the pilot results. Finally, the pilot resulted in changes in citation management tools.

3.3. Data extraction

The data extraction was performed by reading the complete text of all the selected (final selection based on full text) papers and coding the relevant excerpts. Integrated coding was selected as data extraction strategy [8]. Integrated coding includes having a start list of codes as well as creating new codes if necessary (ground-up). It provided the focus for research questions but flexibility regarding findings. Deductive coding would have been too restrictive, and inductive coding might have caused too much bias. Integrated coding made it possible to create a sample list of code categories:

- Why is the metric used?
- What is the effect of metric use?
- Metric
- Importance of the metric
- Context

The coding started by reading the full text and marking relevant quotes with a temporary code. After reading the full text, the first author checked each quote and coded again with an appropriate code based on the built understanding.

In weekly meetings, all authors iteratively built a rule set for collecting metrics and discussed borderline cases. The final rule set was as follows:

- Collect metric if team or company uses it.
- Collect metric only if something is said about why it is used, what effects it causes, or if it is described as important.
- Do not collect metrics that are only used for the comparison and selection of development methods.
- Do not collect metrics that are primarily used to compare teams. (There were cases where a researcher or management uses a metric to compare teams. We wanted to find metrics a team could use.)

Atlas.ti[2] version 7 was used to collect and synthesise the qualitative data. The amount of coded quotes per code can be seen in Table 2. To evaluate the repeatability of finding the same metrics, the second author coded the metrics from three primary studies. The capture-recapture method [34] was then used, which showed that 90% of metrics were found.

Table 2: The amount of found quotes

Code	Amount of quotations
Why is the metric used?	151
What is the effect of metric use?	61
Metrics	102
Importance related to the metric	45
Context	158

A quality assessment form adopted from [9] was used to evaluate the quality of each primary study. A detailed list of quality assessment questions can be found in Appendix C. Additionally, a relevancy factor was added to the same assessment to describe how useful a primary study was for this study. The relevancy factor was evaluated subjectively by the researcher. The scale for the factor is:

- 0 = does not contain any information regarding metrics and should already be excluded
- 1 = only descriptions of metrics with no additional info
- 2 = some useful information related to metrics
- 3 = a good amount of relevant information regarding metrics and metric use

[2]http://atlasti.com/product/features/

Figure 1: Example of qualitative analyses. Reasons and effects of using metrics category "Progress tracking" was formed by organising descriptive codes (boxes in the figure) into a group based on their similarity.

3.4. Data analysis and synthesis

The results of the initial coding of the metrics and other quotes were further synthesised by similarity based categorisation. Metrics codes were grouped based on similarity to enable the categorisation in Table 7 and Table 10. For example, burndown is grouped under velocity and faults per iteration is grouped under defect count. For the reasons and effects, the data synthesis followed the steps recommended by Cruzes and Dybå [8]. The process started by going through all quotes within one code and describing each quote with a more descriptive high level code. Then the high level codes were organised in groups based on their similarity, see, e.g., Figure 1. These groups were then given names, which are seen as categories in our results, for example, see Table 9.

4. Results

This section presents the results of the systematic literature review and provides the answers to the research questions. Section 4.1 describes the overview of studies. Section 4.2 describes the results of the quality evaluation of the primary studies. Section 4.3 presents the found metrics (RQ1), categorises them

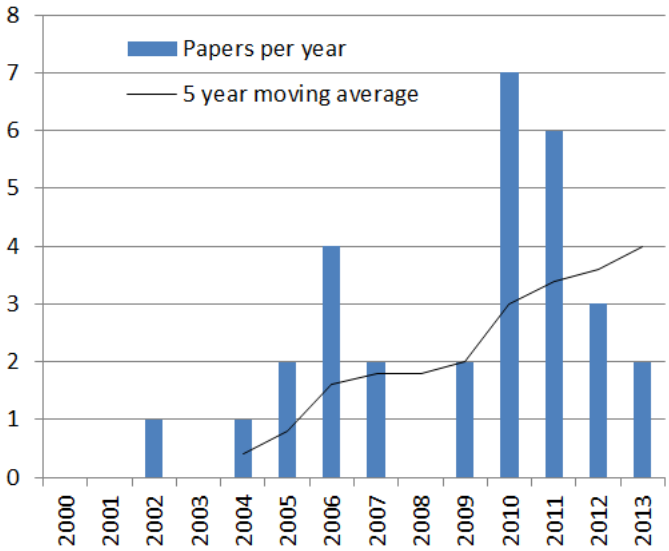

Figure 2: Number of papers per year

based on the entity that is measured, and compares them with metrics suggested by the Agile literature. Section 4.4 describes the reasons for using metrics and also describes the effects of metric use (RQ2). Originally, we tried to separate the RQ2 into two separate sections. However, this resulted in a large amount of repetitive text, as the primary studies often did not make a clear distinction between the motivation for using the metrics (why) and the effects the metrics had. Finally, Section 4.5 describes important metrics (RQ3) by statements from the primary studies as well as by the amount of evidence from the primary studies.

4.1. Overview of studies

This section gives an overview of the primary studies. Table 3 shows the distribution of the primary studies by publication channel. Table 4 lists the primary studies by context factors. Figure 2 shows the number of papers per year.

The study identified 30 primary studies with 36 cases in total. The primary studies were published in 12 different journals, conferences, or workshops, see Table 3. A large share of the primary studies (43%) was published in the Agile Conference. The rest of the studies were published in a wide range of journals, conferences, and workshops.

The primary studies and their context information can be seen in Table 4. For collecting context, we tried to apply Petersen and Wohlin [26] as much as possible. However, the information formed a sparse matrix. Therefore, the summary of our context variables in Table 4 seems quite thin. Furthermore, the

Table 3: The publication distribution of the primary studies

Publication channel	Type	#	%
Agile Conference	Conference	9	43
HICCS	Conference	3	14
ICSE SDG	Workshop	2	10
XP Conference	Conference	2	10
Agile Development Conference	Conference	1	5
APSEC	Conference	1	5
ASWEC	Conference	1	5
ECIS	Conference	1	5
Elektronika ir Elektrotechnika	Journal	1	5
Empirical Software Engineering	Journal	1	5
EUROMICRO	Conference	1	5
ICSE	Conference	1	5
IST	Journal	1	5
IJPQM	Journal	1	5
JSS	Journal	1	5
PROFES	Conference	1	5
Software - Prac. and Exp.	Journal	1	5
WETSoM	Workshop	1	5

industry domain was extracted from the primary studies as it was stated in the papers. Because we do not have other knowledge of the reported cases in the papers than what is written in the papers, it would have been very difficult to match the case context to some specific pre-defined classification. Furthermore, the research method classification is also based on how the method was described in the primary studies. The empirical reports by the case participants that did not describe a proper research method were classified as experience reports.

The earliest study is from 2002, and the rest of the studies are quite evenly distributed from 2002 to 2013. Single-case was the most used research method (60%), followed by experience report (23%), multi-case (10%), and survey (7%).

The Agile method for the studies was identified based on the assessment of the researcher. A specific method was chosen if it seemed to be a primary method in the case. Based on the results, Scrum was the most used Agile method (35%) in the primary studies. XP was the second most used Agile method (20%), while LeanSD (Lean Software Development) and Kanban were used in 5% of the cases. In 33% of the cases, the used Agile method was unclear, and this is marked by "NA" . We decided to include such cases as often in practice a custom or hybrid agile process is used [37].

If it was unclear what Agile method was used, then "NA" was set as the Agile method.

Telecom was the most represented domain (28%), enterprise information systems was the second (19%), and web applications was the third (11%). Forty-

two percent of the cases were of other domains or without domain information.

Table 4: Overview of the primary studies

ID	Year	Resear. meth.	Agile method	Team size	Domain
[S1]	2010	Survey	NA	NA	NA
[S2]	2005	Experience r.	MSF v4.0[3]	NA	NA
[S3]	2009	Multi-case	NA/Scrum/ Scrum	2-10/2-7/4-8	ERP/Graphic design plug-in/Facility management
[S4]	2013	Experience r.	Scrum	25 teams	Software for oil and gas industry
[S5]	2005	Single-case	XP	15	Enterprise information system
[S6]	2002	Experience r.	XP	50	Enterprise resource solution for the leasing industry
[S7]	2011	Survey	Scrum	26 teams	Desktop and SaaS products
[S8]	2010	Experience r.	Scrum	5-9	NA
[S9]	2006	Single-case	XP	15-20	Broadband order system
[S10]	2004	Multi-case	XP/Scrum	4-18/6-9	b-2-b e-commerce solutions/Criminal justice system development
[S11]	2007	Single-case	Scrum	500	Security services
[S12]	2010	Single-case	Scrum	NA	E-commerce
[S13]	2011	Single-case	LeanSD	5±2	Information and communication software development
[S14]	2012	Experience r.	XP	NA	Web application development
[S15]	2006	Multi-case	NA/NA/ NA/NA	2-5/12-15/1-10/6-7	NA/NA/NA/NA
[S16]	2012	Single-case	Scrum	6-8	Web page development
[S17]	2007	Single-case	NA	Comp. 160 devs	Various
[S18]	2010	Single-case	LeanSD	Dev site 600	Telecom
[S19]	2010	Single-case	XP	6-7	Telecom
[S20]	2010	Single-case	NA	NA	Telecom
[S21]	2011	Single-case	Scrum	Dev site 500	Telecom
[S22]	2012	Single-case	NA	NA	Telecom

[3]Microsoft Solutions Framework v4.0

[S23]	2011	Experience r.	Scrum / Kanban	9 and 6	Casino games
[S24]	2011	Single-case	Kanban	6-8	Telecom maintenance
[S25]	2010	Single-case	NA	project size 100	Telecom
[S26]	2011	Single-case	NA	project size 200	Telecom
[S27]	2006	Single-case	XP	15	Enterprise information system
[S28]	2009	Single-case	XP	15	Enterprise information system
[S29]	2006	Experience r.	NA	NA	Telecom
[S30]	2013	Single-case	NA	5	Space mission control software

4.2. Quality evaluation of the primary studies

The quality evaluation was done by the researcher after the data extraction of each primary study. Each category was evaluated on a scale from 0 to 1. The evaluation form was adopted from [9]. The detailed list of quality evaluation questions can be found in Appendix C. Additionally, a relevancy factor was assigned to each study describing its relevancy to this study. The scale for the relevancy can be found in section 3.3.

The perceived quality of the studies varied a great deal (from 0 to 10). Even though there were many low scoring studies, they were included since they still provided valuable insight. For example, in some cases an experience report [S4] provided more valuable data than a high scoring research paper [S25].

According to the quality evaluation, control group and reflexivity had the lowest total scores, while values for research, context, and findings scored the highest. Forty-three percent of the primary studies had a total score of 8, 9, or 10, and 37% of the primary studies had a total score of 1, 2, or 3.

4.3. RQ1: Metrics

RQ1 was *What metrics are used in industrial Lean and Agile software development?* Here we look into the results of this RQ by listing, categorising, and comparing the metrics. All the found metrics are listed by primary study in Table 6. A total of 102 metrics were found in the primary studies. Definitions of metrics can be found in Appendix D. Metrics were only collected if their *reason for use*, *effect of use*, or *importance* was described.

Table 6 provides the raw results of RQ1 but – as it does not provide the necessary high level overview – we also present the metrics under two different categorisation. First, the metrics are presented with the categorisation by Fenton and Pfleeger [10]. We use their categorisation because their work on software metrics is very widely known in the software engineering community (over 4,500 citations in Google Scholar). Second, we categorise and contrast the

Table 5: Quality evaluation of the primary studies

Study	Research	Aim	Context	R.design	Sampling	Ctrl. grp	Data coll.	Data anal.	Reflex.	Findings	Value	Total	Relevancy
[S1]	1	1	1	1	1	0	1	1	1	1	1	10	2
[S2]	0	0	0	0	1	0	0	0	0	0	1	2	2
[S3]	1	1	0	1	0	0	0	0	0	0	0	3	2
[S4]	0	0	0	0	1	0	0	0	0	0	1	2	3
[S5]	1	1	1	1	1	0	1	1	0	1	1	9	3
[S6]	0	0	1	0	1	0	0	0	0	0	1	3	2
[S7]	0	0	0	0	0	1	1	1	0	1	1	5	2
[S8]	0	0	0	0	0	1	0	0	0	1	1	3	3
[S9]	1	1	1	1	0	0	1	1	1	1	0	8	2
[S10]	0	0	1	0	1	1	0	0	0	1	1	5	2
[S11]	0	0	1	0	0	0	0	0	0	1	1	3	3
[S12]	0	0	1	0	0	0	0	0	0	0	0	1	3
[S13]	0	0	0	0	0	1	0	0	0	1	1	3	3
[S14]	0	0	0	0	0	0	0	0	0	0	0	0	2
[S15]	1	1	0	1	1	1	1	1	0	1	1	9	2
[S16]	1	0	1	0	1	0	0	0	0	0	0	3	2
[S17]	1	1	1	1	1	0	1	0	0	1	1	8	3
[S18]	1	1	1	1	1	0	1	1	0	1	1	9	3
[S19]	1	1	1	1	1	0	1	1	1	1	1	10	2
[S20]	1	1	0	1	0	0	0	0	0	1	0	4	2
[S21]	1	1	1	1	1	0	1	1	1	1	1	10	2
[S22]	1	1	1	1	1	0	1	1	1	1	1	10	2
[S23]	0	0	1	0	0	1	0	0	0	1	1	4	2
[S24]	0	0	1	0	1	0	0	0	0	1	1	4	2
[S25]	1	1	1	1	1	0	1	1	1	1	1	10	3
[S26]	1	1	1	1	0	0	1	1	1	1	1	9	2
[S27]	1	1	1	1	1	0	1	1	0	1	1	9	2
[S28]	1	1	1	1	1	0	1	1	0	1	1	9	3
[S29]	0	0	0	0	0	0	0	0	0	0	1	1	3
[S30]	0	0	1	0	1	0	0	0	0	0	1	3	2
Total	16	15	20	15	18	6	14	13	7	21	24		

SLR metrics to the metrics suggested in the original works on Agile methods, i.e. Scrum [33], XP [2], Kanban [1], and LeanSD [27]. This comparison allows us to see whether practitioners follow the metrics suggested in the Agile methods or not.

The categorisation by Fenton and Pfleeger [10] has two dimensions: entities and attributes. The entities tell whether the metrics are related to either, processes that are "collection of software related artifacts", products that are "artifacts, deliverables, or documents that result from a process activity", or resources that are "entities required by process activity". The attributes distinguish between internal and external attributes. Internal attributes "can be measured by examining the product, process, or resource on its own", whereas external attributes "can be measured only with respect to its environment".

Table 7 shows that metrics were largely applied to products, test plans, code, builds, features, requirements, and defects. Most of the entities in the Products class were measured internally, except the products entity, which was measured mostly externally. Furthermore, testing, implementation, and the whole development cycle were measured mostly internally in the Processes class. Only two metrics are related to measuring the Resources class.

In Table 7 the same metric can exist in different places depending on how it is applied. Categorising the metrics according to Fenton and Pfleeger [10] is not a trivial task, as metrics can be categorised into any of several classes depending on the viewpoint. For example, defect count can be an internal process measure of software testing, but at the same time the number of defects experienced by the customer can be an external measure of the software product or even customer satisfaction as the software is used by the customer. We categorised the metrics based on their application in the original sources, although it may make Table 7 look inconsistent.

Table 8 compares the SLR metrics with the ones suggested in the Agile methods, i.e. the ones presenting Scrum [33], XP [2], Kanban [1] and LeanSD [27]. The rightmost column in Table 8 describes the Agile method used in the primary studies ('S'=Scrum, 'X'=XP, 'K'=Kanban, 'L'=LeanSD) . In some primary studies, it was hard to identify a specific Agile method, thus 'NA' is used to describe those cases. The number before the primary study reference defines the index of the metric in the list of metrics for the study in question in Table 6.

Table 8 shows that he most popular metrics suggested by the Agile literature are Effort estimate and Velocity. This indicates the need to track progress. In addition, metrics related to quality and testing had some importance, as measuring initial quality and the quantity of unit and acceptance tests had some popularity. Actual development time (XP), load factor (XP), Due date performance (Kanban), Issues and blocked work items (Kanban), and Flow efficiency (Kanban) were not described in any primary studies. It is difficult to say why these metrics had not been used at all. Perhaps they are not needed, or the issues they measure are already covered by existing metrics, e.g. actual development time can be in a practitioner's view quite similar to velocity.

Table 8 shows that many metrics (39%) found in the primary studies were

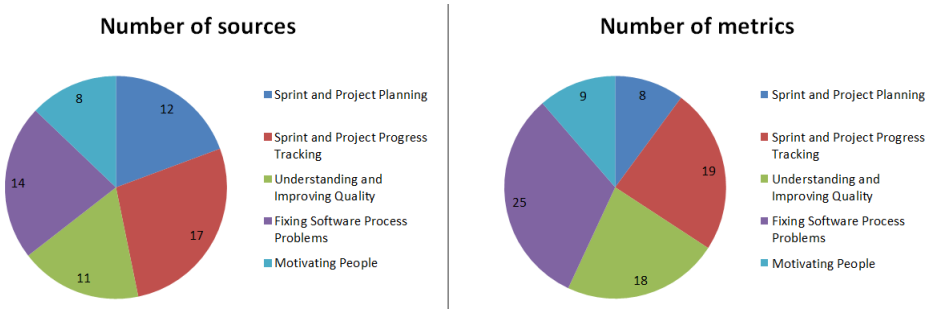

Figure 3: Number of sources and metrics for the reasons for and effect of using metrics

not suggested in the original works on Agile methods. Thus, it appears that practitioners add and invent new metrics according to their needs. Additionally, agile literature cannot possibly suggest all metrics that could be compatible with Agile values, for example, a Business value delivered metric focuses on the customer whose satisfaction is a top priority in Agile software development. However, the Agile literature studied in Table 8 does not suggest measures for business value. Furthermore, Agile literature is missing some basic metrics of software engineering such as the defect count. We could argue that since agile emphasises progress as working code, then defect count metric is not needed as by definition working code cannot include defects. However, many of the cases still measured the defect count.

4.4. RQ2: Reasons and effects of using metrics

RQ2 was *What are the reasons for and effects of using metrics in industrial Lean and Agile software development?*

We divided the reasons and effects of using metrics to five categories: Sprint and Project Planning, Sprint and Project Progress Tracking, Understanding and Improving Quality, Fixing Software Process Problems, and Motivating People. These categories are described in Sections 4.4.1 to 4.4.5, respectively. Table 9 lists the primary studies that have contributed to each of the categories and Figure 3 illustrates the number of papers and number of metrics in the different catogries. We would like to highlight that a single metric can belong to several categories depending on how it is used. Thus, the contents of our five categories are heavily dependent on how using the metrics was described in the primary studies.

4.4.1. Sprint and Project Planning

From the primary studies we found three types of planning activities: prioritisation, scoping, and resourcing. **Prioritization** of tasks was one of the main activities metrics were used for. At Objectnet, effort estimates were used to prioritise the features for the next release and as basis for resourcing [S9]. Teams at Adobe Systems used effort estimates to prioritise activities based on

Table 6: RQ1: Metrics by the primary studies

ID	Metrics
[S1]	Business value delivered, customer satisfaction, defect count after testing, number of test cases, running tested features
[S2]	Velocity, Work in progress
[S3]	Critical defects sent by customer, open defects, test failure rate, test success rate, remaining task effort, team effectiveness
[S4]	Technical debt board, build status, technical debt in effort
[S5]	Burndown, check-ins per day, number of automated passing test steps, faults per iteration
[S6]	Velocity, story estimates
[S7]	Burndown, story points, # of open defects, # of defects found in system test, defects deferred, Net Promoter Score
[S8]	Story points, task effort, velocity, operations' velocity
[S9]	Effort estimate
[S10]	# of defects/velocity
[S11]	Revenue per customer
[S12]	Task's expected end date, effort estimate, completed web pages, task done
[S13]	Fix time of failed build, story flow percentage, percentage of stories prepared for sprint, velocity of elaborating features, velocity of implementing features
[S14]	Build status, test coverage, test growth ratio, violations of static code analysis, # of unit tests
[S15]	Effort estimate
[S16]	Sprint burndown, release burndown, cost performance index, schedule performance index, planned velocity
[S17]	Common tempo time, number of bounce backs, cycle time, work in progress, customer satisfaction (Kano analysis), effort estimate kits
[S18]	Lead time, processing time, queue time
[S19]	Change requests per requirement, fault slips, implemented vs wasted requirements, maintenance effort, lead time
[S20]	Number of requests from customers, inventory of requirements over time
[S21]	Rate of requirements per phase, variance in handovers, requirement's cost types
[S22]	# of requirements per phase, lead time
[S23]	Average velocity / work in progress, cycle time, pseudo velocity
[S24]	Lead time, work in progress
[S25]	Defect trend indicator, # of defects in backlog, predicted # of defects
[S26]	Throughput, queue
[S27]	Burndown, check-ins per day, number of automated passing test steps, number of new and open defects
[S28]	Burndown, number of automated passing test steps, check-ins per day
[S29]	Story estimate, story complete percentage
[S30]	Progress as working code

Table 7: Metric categorisation based on [10]

Entities	Attributes	
Products	**Internal**	**External**
Products	Running tested features [S1], build status [S4, S14]	Customer satisfaction [S1, S3, S7, S17, S19, S20], progress as working code [S30]
Test plans	Number of test cases [S1]	
Code	Technical debt in categories [S4], technical debt in effort [S4], violations of static code analysis [S14]	
Features	task's expected end date [S12], task done [S12], effort estimate [S7, S8, S8, S9, S12, S15, S15, S15, S15, S17, S29], story complete percentage [S29]	Business value delivered [S1]
Requirements	Requirement's cost types [S21], Percentage of stories prepared for sprint [S13]	
Defects		Defect trend indicator [S25], predicted number of defects [S25]
Processes		
Testing	defect count [S1, S3, S5, S7 ,S7, S10, S25, S27], test success rate [S3], test failure rate [S3], defects deferred [S7], test coverage [S14], test growth ratio [S14]	Number of bounce backs [S17], fault slips [S19]
Implementation	Velocity [S1, S2, S3, S5, S6, S8, S8, S10, S13, S16, S16, S16, S23, S27, S28], number of unit tests [S1, S5, S14, S27, S28], completed web pages [S12], cost performance index [S16], schedule performance index [S16], planned velocity [S16], common tempo time [S17], check-ins per day [S5, S27, S28], fix time of failed build [S13]	Story flow percentage [S13]
Requirements engineering	velocity of elaborating features [S13]	
Whole development cycle	cycle time [S17, S23], lead time [S18, S19, S22, S24], processing time [S18], queue time [S18], maintenance effort [S19], work in progress [S2, S17, S20, S21, S22, S23, S24], variance in handovers [S21], throughput [S26], queue [S26], implemented vs wasted requirements [S19]	
Resources		
Team		Team effectiveness [S3]
Customer	Revenue per customer [S11]	

Table 8: Metrics from the Agile literature compared to the metrics found in this study (Number prior to study reference is the index in the list of metrics for the study in question in Table 6; The characters refer to the methodologies; S=Scrum, X=XP, L=LeanSD, K=Kanban, NA=unclear)

Metrics suggested	**Method**	**SLR Sources**
Effort estimate	Scrum, XP, LeanSD	S: 2[S7], 1[S8], 2[S8], 3[S12]. X: 1[S9]. NA: 1[S15], 6[S17], 1[S29].
Velocity (Includes total work remaining from Scrum and effort left from Scrum and XP.)	Scrum, XP, LeanSD	S: 3[S8], 2[S10], 4[S8], 1[S16], 2[S16], 5[S16], 1[S23]. X: 1[S6], 1[S5], 1[S27], 1[S28]. K:4[S23]. L: 5[S13]. NA:1[S2], 5[S3].
Written and passed unit tests	XP, LeanSD	S: 3[S5], 3[S27], 2[S28], 5[S14]. NA: 5[S1].
Actual development time	XP	
Load factor	XP	
Work in progress	Kanban	S:1[S21]. K:2[S23], 2[S24]. NA:4[S17], 2[S20], 1[S22].
Lead time	Kanban	X:5[S19]. K: 1[S24]. L:1[S18]. NA:2[S22].
Due date performance	Kanban	
Throughput	Kanban	NA:1[S26].
Issues and blocked work items	Kanban	
Flow effiency	Kanban	
Initial quality	Kanban, LeanSD	S: 3[S7], 4[S7]. X:4[S5], 1[S10], 4[S27]. NA:3[S1], 2[S3], 2[S25].
Failure load	Kanban	NA:2[S17].
Cycle time	LeanSD	K:3[S23]. NA:3[S17].
Value Stream Maps (Work time, wait time)	LeanSD	L:2[S18], 3[S18].
Number of written and passed acceptance tests per iteration	LeanSD	NA:4[S1], 3[S3], 4[S3].
Metrics not suggested in Agile methods, but used in the primary studies		S: 1[S4], 2[S4], 3[S4], 4[S7], 5[S7], 1[S11], 1[S12], 3[S12], 4[S12], 3[S16], 4[S16], 2[S21], 3[S21]. X: 2[S5], 2[S27], 3[S28], 1[S14], 2[S14], 3[S14], 4[S14], 1[S19], 2[S19], 3[S19], 4[S19]. L: 1[S13], 2[S13], 3[S13], 4[S13]. NA:1[S1], 2[S1], 3[S1], 1[S3], 6[S3], 1[S17], 5[S17], 1[S20], 1[S25], 3[S25], 2[S26], 2[S29], 1[S30].

Table 9: Reasons and effect of using metrics by sources

Categories	Sources and the metrics
Sprint and Project Planning	Velocity [S2, S16, S23], Effort Estimate [S6, S8, S9, S12, S29], Value to customer [S8, S11, S17], Lead time [S23, S24], Task done/undone [S12], Task's expected done date [S12], Predicted N of defects [S25], Skills needed [S17]
Sprint and Project Progress Tracking	Completed work (web-pages [S12], task kits [S17]), N of automated passing tests [S5, S27, S28], Burn-down [S5, S7, S16, S27, S28], Check-ins [S5, S27, S28], Defects [S25, S27], Defect trends [S25], Story percent complete [S29], Cost types [S21], Rate of requirements per phase [S21], Variance in handovers [S21], Technical debt board [S4], Cycle time[S23], Common tempo time [S17], Work in Progress[S2], Story flow percentage[S13], Team effectiveness [S3], Inventory of requirements over time [S20], Effort estimate [S8, S23], N of requirements per phase [S22]
Understanding and Improving Quality	N of change requests[S19, S22], Maintenance effort[S19], Net Promoter Score[S7], Defects[S5, S7, S10], Defect deferred [S7], Critical defects sent by customers [S3] Burn-down [S28], Check-ins[S28], N of automated passing tests [S5, S28], Build status, N of unit tests [S14], Test coverage [S14], Test growth ratio [S14], Violation of static code analysis [S14], Technical debt board [S4], Work in progress [S17], Story percent complete [S29], Cycle time[S17]
Fixing Software Process Problems	Lead time [S18, S26], Processing time [S18], Queue time [S18, S26], Cost types [S21], Rate of requirements per phase [S21], Variance in handovers [S21], N of requirements per phase [S22], Story flow percentage[S13], Defect trend [S25], Cost performance index [S16], Schedule performance index [S16], Story percent complete [S29], Work in Progress [S2, S17], Inventory of requirements over time [S20], Velocity [S8, S10], Burndown [S28], percentage of stories prepared for sprint [S13], N of bounce backs [S17], N of automated passing tests [S27], Burn-down [S27], Check-ins [S27], Defects [S27], N of work items [S20, S22], Fix times of failed build [S13], Violation of static code analysis [S14]
Motivating People	Defects [S3], Defect trend [S25] Fix times of failed build [S13], Build status [S4, S14], Violation of static code analysis [S14], Technical debt board [S4], N of automated passing tests [S5], Work in Progress [S17], Velocity [S6]

relative value and relative effort [S8]. At Timberline Inc, they used Kano analysis as a 'voice of customer' so that prioritisation decisions could be based on facts instead of political power [S17]. Product owners at WMS Gaming used lead time to schedule high priority features and plan demo dates with customers [S23]. Similarly, at Verisign Managed Security Services, they used the revenue per customer metric to allow higher valued features to be prioritised higher in the backlog [S11].

Scoping. Metrics were used to estimate the size and number of features that could be taken under development. Velocity was used to improve effort estimates for the next planning session, which helped to estimate the scope of the next iteration [S16]. The Scrum master and product owner at a Korean e-commerce company used estimates to check if the planned scope would be possible to complete during the next iteration [S12]. At WMS Gaming, they used pseudo-velocity and average velocity to plan their releases [S23]. Additionally, the Velocity / 2 metric was used as a scoping tool for a release [S23]. The team had enough work not to sit idle, but there was still enough time to fix high priority defects. In Ericsson's product maintenance team, lead time was used to understand whether all planned corrections could be completed before the release date [S24]. At Avaya Communications, they used story estimates to predict the iteration where a feature would be completed [S29].

Furthermore, velocity was used to define a minimum delivery level for the iteration where "must have" requirements are assigned, and a stretch goal where lower priority requirements are assigned [S2]. At a Korean e-commerce company, they marked tasks done and undone, which made it possible to move undone tasks to the next iteration [S12]. The expected date of task completion metric was used so that other team members could plan their own work and avoid idle time [S12]. For example, a developer could know when she could start implementation because the designer had informed her of the expected date of completion for the design

Resourcing. Metrics were used for resourcing decisions and development flexibility. At Timberline Inc, they broke down requirements into smaller pieces that were estimated in effort to understand what skills are needed to complete the work [S17]. At ThoughtWorks, stories were used to break down new functionality, and effort estimates of the stories were summed to understand the needed resources [S6]. At Ericsson, the predicted number of defects was used to plan the removal of defects [S25]. If the removal of defects were not well planned, it could cause delays for the release and thus increase costs for the project.

4.4.2. Sprint and Project Progress Tracking

Another prominent reason for metric use in the primary studies was progress tracking. The reasons for using metrics in progress tracking are divided into project progress, increasing visibility, achieving goals, and balancing workflow.

Project progress. Metrics were used to monitor the progress of the project. The completed web pages metric was used as a measure of progress at a Korean e-commerce company. The number of automated passing test steps was used as a

measure of progress in terms of completed work at Mamdas [S5]. At Timberline Inc, breaking down tasks to 'kits' between two to five days enabled progress monitoring [S17]. A set of metrics (burndown, check-ins per day, number of automated passing test steps, number of new and open defects) was developed to manage risks and provide timely progress monitoring [S27]. Developers at Avaya Communications used the story percent complete metric to give an assessment of progress [S29]. However, a team at NASA Ames Reserch Center did not want to spend resources on estimating features, and instead they focused their efforts on developing software [S30]. Every six weeks they demonstrated their progress to the customer with working code.

Metrics were also used to give a higher level of understanding about progress. The release burndown showed project trends and could be used to predict the completion date [S16]. Also, the release burndown could reflect addition or removal of stories. At Ericsson, cost types, rate of requirements over phases, and variance in handovers were used to provide overview of progress [S21]. Metrics (burndown, check-ins per day, number of automated passing test steps) were used to communicate progress to upper management [S5], ensure good progress to external observers, and ensure that key risks were under control [S5,S27,S28].

Increasing visibility. Metrics were used to simplify complex aspects of software development and increase visibility for all stakeholders. Cost types, rate of requirements over phases, and variance in handovers were used to increase the transparency of end-to-end flow in a complex system [S21]. Similarly at Petrobras, the technical debt board was used to make technical debt issues visible and easier to manage [S4]. To replace individual perception with facts, burndown, check-ins per day, number of automated passing test steps, number of open and new defects metrics were used [S27].

Furthermore, metrics were used to keep the team informed. At Ericsson, the defect trend indicator was used to monitor the defect backlog and spread the information to project members [S25]. At WMS Gaming, a cycle time metric was used to let the team track their performance [S23]. At Avaya Communications, story percent complete metrics were generated automatically when tests were run and thus kept everyone on the same page and eliminated schedule surprises [S29]. Additionally, the metric results were required to be reported periodically.

Achieving goals. Metrics were used to understand whether project goals could be achieved and to cut down the scope of an iteration or to add more resources if it did not seem that all tasks could be completed at the current pace. At Timberline Inc, there was a need for a simple indicator that would quickly tell whether a project was under control [S17]. They used common tempo time to understand if the project was on target for delivery. Furthermore, if common tempo time indicated too much planned work, then the tasks would be cut or more resources would be added [S17]. Similarly, employees were trained with multiple skills, e.g., customer support did testing and documentation, engineers were taught how to input their material into the system, so in case of an imbalanced workload the work could be reorganised to achieve a more balanced workflow. At Microsoft Corporation, they monitored work in progress to predict lead time, which in turn would predict a project schedule [S2]. At Adobe Sys-

tems, sprint burndown was used to tell the team if they were on track regarding the sprint commitments [S7]. Similarly at Mamdas, component level burndown was used to notice that a component was behind schedule, so resources were added and scope was reduced for the release [S5]. Burndown was also used to mitigate the risk of developers spending too much time perfecting features rather than finishing all the tasks of the iteration [S28]. Furthermore, at a Slovenian publishing company, the release burndown made the correlation clear between work remaining and the team's progress in reducing it, and when the release burndown showed that work remaining was not decreasing fast enough, so the scope of the release was decreased [S16]. Story flow percentage was used so that a developer could finish a story in a steady flow [S13]. A story implementation flow metric describes how efficiently a developer has been able to complete a story compared with the estimate. Similarly, if team effectiveness was not high enough to complete tasks, resources from other teams can be used [S3]. Other actions that were suggested in case of low team effectiveness were the reduction of tasks and working overtime.

Balancing workflow. Metrics were used to balance workflow to prevent overloading people. At Ericsson, inventory of requirements over time was used to identify large handovers of requirements that would cause overloading situations to employees [S20]. The aim was to have a steady flow of requirements. Similarly at Citrix Online, the operations department was overloaded so they decided to start evaluating incoming work with Ops story points to level the workload [S8]. Moreover, people should be respected by having a balanced workload to avoid overload situations [S22]. This could be achieved by measuring the number of requirements per phase, which would reveal the peaks of the workload. Timberline Inc tried to pace work according to customer demand [S17]. However, too much work was pushed to development, which caused many problems, including developers feeling overworked. They started using common tempo time to make sure there would be balance of workflow.

At Ericsson, variance in handovers was used to guarantee that requirements would flow evenly [S21]. Mamdas was measuring check-ins per day metric, which measured how often code was committed to the main trunk [S5]. The point was to keep people from committing only at the end of the iteration, and instead verify that work was spread evenly across iterations. At WMS Gaming, they had problems with large tasks blocking other work, so they set a rule that only tasks of a certain size (8 story points) could be taken for development [S23].

4.4.3. Understanding and Improving Quality

This section describes how metrics were used to understand the quality of the product both before and after release. This section is divided into three parts. The first two describe how quality was understood through metrics and how it was improved. The last part describes how metrics were used to ensure that the product is tested thoroughly, which is a key part in getting information of software quality.

Understand the level of quality. Metrics were used to understand the level of quality after the release. The number of change requests from the

customer was used as an indicator of customer satisfaction [S19]. Maintenance effort was used as an indicator of overall quality of the released product [S19]. Number of maintenance requests was used as an indicator of built-in quality [S22].

Metrics were also used to understand the level of quality before the release. At Adobe Systems, they measured pre-release quality with Net Promoter Score which was measured from pre-release customer surveys [S7]. Net Promoter Score measures how willing a customer is to recommend the product to another potential customer. They also measured defects found in the system test that was used to measure the quality of software delivered to the system test process. Additionally, they measured defects deferred, which was used to predict the quality customers would experience. Defects deferred were defined as the defects that are known but are not fixed for a release, usually due to time constraints. At Mamdas, faults per iteration were used to measure the quality of the product [S5]. At Escrow.com, number of defects was used to delay a release when too many defects were noticed in a QA cycle [S10].

Increase quality. Metrics were used to increase the level of quality. Governance mechanisms, which included a set of metrics (burndown, check-ins per day, and number of automated passing test steps), were used to increase product quality [S28]. At T-Systems International, the quality manager used a set of metrics (build status, number of unit tests, test coverage, test growth ratio, violations of static code analysis) to improve the internal software quality of the project [S14]. Build status was measured to prevent defects reaching production environment. Similarly, the violations of static code analysis metric was used to prevent critical violations. Furthermore, critical defects sent by customers were tracked and fixed to prevent losing customers [S3]. Finally, technical debt board was used to reduce technical debt [S4].

Ensure the level of testing. Metrics were used to make sure the product was tested thoroughly. At T-Systems International, test coverage was used to evaluate how well the code was tested [S14]. However, in Brown-field (legacy) projects it was better to measure test-growth-ratio since there might not be many tests in the existing code base. At Timberline Inc, work in progress was measured so it could be minimised [S17]. A large amount of work in progress would contain many unidentified defects, which would be discovered eventually. At Mamdas, using number of automated passing test steps decreased the risk that the product would be unthoroughly tested [S5]. Similarly, the number of automated passing test steps was used to make sure regression tests were ran and passed every iteration. Finally, the story percent complete metric supported test driven development by requiring unit tests to be written for progress tracking [S29]. Metrics were also used to react to test information. At Timberline Inc, monitoring cycle times revealed high time consumption on manual testing [S17]. The cause was an unmotivated person who was then moved to writing automated test scripts, which he preferred over manual testing. When the number of written and passed unit tests was not increasing, an alarm was raised at Mamdas [S28]. The issue was discussed in a reflection meeting in which they understood that too much work was being put into a single tester writing the tests,

and once she was doing work for another project, no tests were written. The team then started to learn to write the tests themselves, and later, a dedicated tester was assigned to write the tests.

4.4.4. Fixing Software Process Problems

In several studies, software metrics helped to understand and fix problems in software engineering processes.

At Ericsson, Value Stream Maps (VSM) were used to spot waste in several spots of the development process [S18]. First, lead time, processing time, and queue time metrics were used to identify waste – a requirement would wait for a long time before a full specification [S18]. A solution idea was created where a quick high level proposal would be sent to the customer without the need for an in-depth specification. The customer could then use the high level proposal to evaluate whether they wanted to pursue that requirement further. Second, long processing times for the solution proposal phase indicated a waste of motion where requirements are clarified between the marketing and the development unit. The solution idea was to increase close collaboration between the marketing unit and the development unit, at least for the more complex requirements. Third, there was time wasted of waiting in the design phase, which could be improved by starting real work only when the purchase order was received, not when requests were received. Fourth, lead time, processing time, and queue time metrics were used to identify the waste of waiting in testing phases [S18]. The improvement suggestion was to provide an earlier beta version and to make testing phases parallel. Many of the improvement ideas came from meetings where VSM were used as a base for discussion.

More examples from Ericsson showed that cost types, rate of requirements over phases, and variance in handovers were used to identify bottlenecks [S21]. They noticed that focusing on deadlines caused many requirements to be transferred to the system test phase close to the deadline. The improvement suggestion was to focus more on continuous delivery instead of focusing on market driven deadlines. Furthermore, Kanban was suggested as a development method to accomplish the continuous delivery capabilities. Another case study from Ericsson revealed that cumulative number of work items over time metric was used to identify bottlenecks in the development process [S22]. Throughput and queue time metrics were used to identify a bottleneck in the network integration test phase, which led to using other testing practices in future projects [S26]. Similarly, measuring story flow percentage allowed the identification of waste related to context shifts at Systematic [S13].

Metrics were used to identify problems and find improvement opportunities. Defect trend indicator was used to provide the project manager an ISO/IEC 15939:2007 compatible indicator for problems with the defect backlog [S25]. Basically, the indicator shows whether the defect backlog increases, stays the same or decreases in the coming week. The project manager could then use the information to take necessary actions to avoid possible problems. At a Slovenian publishing company, schedule performance index and cost performance index were used to monitor for deviances in the progress of the project and providing

early signs if something goes wrong [S16]. Developers at Avaya had issues with the 80/20 rule according to which the last 20% of iteration content takes 80% of the time [S29]. With the metrics that their T3 tool provided (e.g story percent complete) they were able to see the early symptoms of various problems that can cause delays, and thus react early. Additionally, monitoring work in progress was used to identify blocked work items and the development phase where the blockage occurred [S2].

Inventory of requirements over time was used to identify problems in the development process [S20]. One improvement suggestion was to change from a push to a pull approach so that the team could adjust the workload to enable a more continuous delivery. Another improvement suggestion was to add intermediate release versions so that integration and testing would happen more often and problems could be identified earlier than close to the actual release. A similar solution was applied at Timberline Inc., where inventory of work in progress (with respect to requirements) was kept low, which meant that design, implementation, and testing could start earlier and problems in requirements would get caught sooner [S17].

Citrix Online started measuring velocity for their operations department as well [S8]. This led to development departments trying to decrease their products' operations story points to enable faster releases. The reduction in story points was made possible by creating hot deployment strategies and providing better documentation.

At an Israel Air Force IT department, Mamdas, they were using burndown to follow their progress [28]. However, when they noticed that work remaining was not decreasing according to remaining resources they had to make changes. In their iteration summary meeting, they decided to pursue senior engineers to help them create optimal development environments and continuous build systems. Also, they decided to evaluate customer requests in more detail to avoid over polishing features.

A team working on automating workflows in a criminal justice system noticed that their velocity estimations were inaccurate, which led to work items being divided into smaller pieces to improve the accuracy of the estimates [S10]. The division of work items meant that the team needed to perform more analysis of the features during planning.

When the story implementation flow metric showed a drop and project managers complained about clarifications about features from the customer were late, a root cause analysis meeting was held [S13]. Also, after starting to use the implementation flow metric new policies were stated to keep the flow high: the percentage of stories prepared for sprint must be 100%, and implementation flow must be at least 60%. Moreover, both of the metrics need to be reported monthly. Root cause analysis was also conducted at Timberline Inc. to decrease the number of bounce backs [S17].

The reasons for the values of metrics (burndown, check-ins per day, number of automated passing test steps, number of new and open defects) were discussed in an iteration summary meeting because it can be hard to analyse metrics without understanding the context [S27]. Similarly at Ericsson, the number of

work items per phase was used to ask a development unit about the values of the metric, and the development unit confirmed that people felt overloaded, as the metric suggested [S20]. Furthermore in another case at Ericsson, if the number of work items was outside the control limits one could discuss the workload with the developers [S22].

At Systematic, after analysing long fix times for broken builds the team added automatic static code analysis checks to code check-in to catch defects earlier [S13]. Similarly at T-Systems International, quality managers could change the coding style guide and code standards based on the results of *violations to static code analysis* metric [S14].

4.4.5. Motivating People

This section describes the motivating effects that the metrics had on people.

Metrics were used to motivate people to react faster to problems. The number of defects was shown in monitors in hallways which motivated developers to fix the defects [S3]. Similarly, total reported defects, test failure rate, and test success rate were also shown throughout the organisation, which motivated people to avoid problems and fix the problems fast. At Systematic, they measured fix time of broken build and showed the time next to the coffee machine. It provoked discussion about the causes of long fix times, and eventually the developers fixed the builds faster [S13]. The metric was later declared mandatory for all projects. Also, the reasons for long fix times were investigated. Similarly at Petrobras, build status was visible just minutes after commits, which helped to create a culture where developers react with high priority to broken builds [S4]. This helped keeping the main branch to be closer to a deployable state at all times. Build status was used to motivate people to fix the build as fast as possible [S4]. Moreover, violations of static code analysis caused developers to immediately fix the issue, because the violations could cause a broken build status [S14]. Additionally, developers could get faster feedback on their work. Furthermore, developers could have more confidence in performing major refactorings with the safety net the violations of static code analysis metric provided.

Metrics were used to change employees' behavior. At Petrobras, they used a technical debt board to discuss technical debt issues in their projects. In the meetings, team members agreed which technical debt issues they would focus on solving until the next meeting [S4]. Additionally, team members sought help from the architecture team to reduce technical debt, e.g., by implementing automatic deployment systems and improving source code unit testability. At Mamdas, measuring the number of automated passing test steps changed the team's behaviour to write more unit tests [S5]. Metrics were also used to prevent harmful behaviour such as cherry picking features that were most interesting to the team [S17]. Measuring work in progress (WIP) and setting WIP limits prevented cherry picking by enforcing working on only two features at a time, and preventing them from working on lower priority but more interesting features. Finally, at Ericsson a defect trend indicator created crisis awareness and motivated the developers to take actions to avoid possible problems [S25].

There can also be negative effects of using metrics. Using a velocity metric had negative effects such as cutting corners in implementing features to maintain velocity with the cost of quality [S6]. For example, the managers excused the developers from writing tests, and the testers cut on the thoroughness of the testing in hopes to maintain the velocity. Similarly, [S14] also hints at dysfunctional use of metrics, for example developers causing a broken build if broken build is used as a KPI (Key Performance Indicator).

4.5. RQ3: High influence metrics

RQ3 was *What metrics have high influence in industrial Lean and Agile software development?* In this section, we highlight the most influential metrics found in our study. We understand that influence is subjective and highly dependent on the specific circumstances. Thus, our list should be taken as descriptive and not as prescriptive. We analyse the high influence metrics from both qualitative and quantitative perspective. Our qualitative approach subjectively measures the perceived importance of each metric based on the reported experiences in the primary studies. We have summarised the results of the qualitative analysis by assessing the perceived importance of each metric from 1 (low) to 3 (high); see Table 10 and Figure 4. The assessment is based on the statements in the articles. Metrics were considered important if the author of the primary study or case employees praised the metric. Also, metrics were considered important if there were signs of continuous use of the metric. Furthermore, if metrics had positive correlation to important output measures such as project success, they were considered important. The qualitative influence evaluation is presented in more detail in section 4.5.1.

Our quantitative approach to high influence metrics is more straightforward. It measures the amount of evidence for using a certain metric by frequency of occurrences in the primary studies. Basically, the amount of evidence is the number of sources that have reported using a certain metric. However, metrics that were only mentioned by name without any reasons for use, effects of use, or importance were not counted. Thus, we considered that simply naming a metric was not enough to provide evidence of its use.

The results of the analysis of high influence metrics are summarised in Table 10. Both the frequency of occurrences and the qualitative perceived importance are presented, and the table is ordered by the sum of ranks divided by two. This means that, as Velocity is ranked number one in both occurrences and perceived importance, it receives a value of one (1+1)/2. Cycle time is ranked as 9th and 24th, so the sum of ranks divided by two is 16.5, making it the lowest ranked metric.

4.5.1. Qualitative Approach—Perceived importance of metrics

In this section we describe the qualitative reasoning that was reported in the primary studies regarding the perceived importance of the metrics.

Progress as working code was considered as one of the cornerstones of Agile [S30]. Story flow percentage and velocity of elaborating features were considered

as key metrics for monitoring projects [S13]. A minimum of 60% for story flow was identified as a key limit. Similarly, velocity for elaborating features should be as fast as velocity of implementing features. They reported that using both aforementioned metrics *"drive behaviors to let teams go twice as fast as they could before"*.

The story percent complete metric was considered valuable since it embraces test driven development—no progress is made before a test is written [S29]. Also, the story percent complete metric was considered more accurate than the previously used metric; however, that metric was not mentioned. The story percent complete metric gave a normalised measure of progress compared to developer comments about progress. The metric leveraged the existing unit testing framework and thus required only minimal overhead to track progress. Furthermore, team members seemed to be extremely happy about using the story percent complete metric.

Practitioners at Ericsson valued the transparency and the overview of progress that the metrics (cost types, rate of requirements over phases, and variance in handovers) were able to provide to the complex product development with parallel activities [S21].

Effort estimates were considered important in release planning, especially in terms of prioritisation [S9]. According to a survey [S7], top performing teams at Adobe Systems estimated backlog items with relative effort estimates. Similarly, pseudo-velocity, which was used by a Kanban team, was considered essential for release planning [S23]. Moreover, burndown was valuable in meeting sprint commitments [S7]. Furthermore, managers said burndown was important in making decisions and managing multiple teams [S5]. However, developers did not consider burndown important [S5]. According to a survey [S1], project success had a significant positive relationship with the following metrics: team velocity, business value delivered, running tested features, defect count after testing, and number of test cases. However, there were no detailed descriptions of these metrics.

In another case at Ericsson, VSMs were used to visualise problem areas and facilitate discussion for possible improvements [S18]. Practitioners valued how the maps were easy to understand. Metrics that were used to build VSMs were lead time, processing time, and queue time. Similarly, technical debt board, which visualised the status of technical debt, was considered important because it gave a high level understanding about the problems [S4]. The board was then used to plan actions to remove the technical debt.

Net Promoter Score, which measures the probability of a customer recommending the product to another potential customer, was said to be *"one of the purest measures of success"* [S7]. Similarly, projects that were said to be definitely successful measured customer satisfaction often or always. Also, the more often customer satisfaction was measured, the more likely it was that the project would have good code quality, and the project would succeed. Similarly, the defects deferred metric was seen as a good predictor of post-release quality because it correlated with issues found by the customers [S7].

Defect prediction metrics *predicted number of defects in backlog* and *defect*

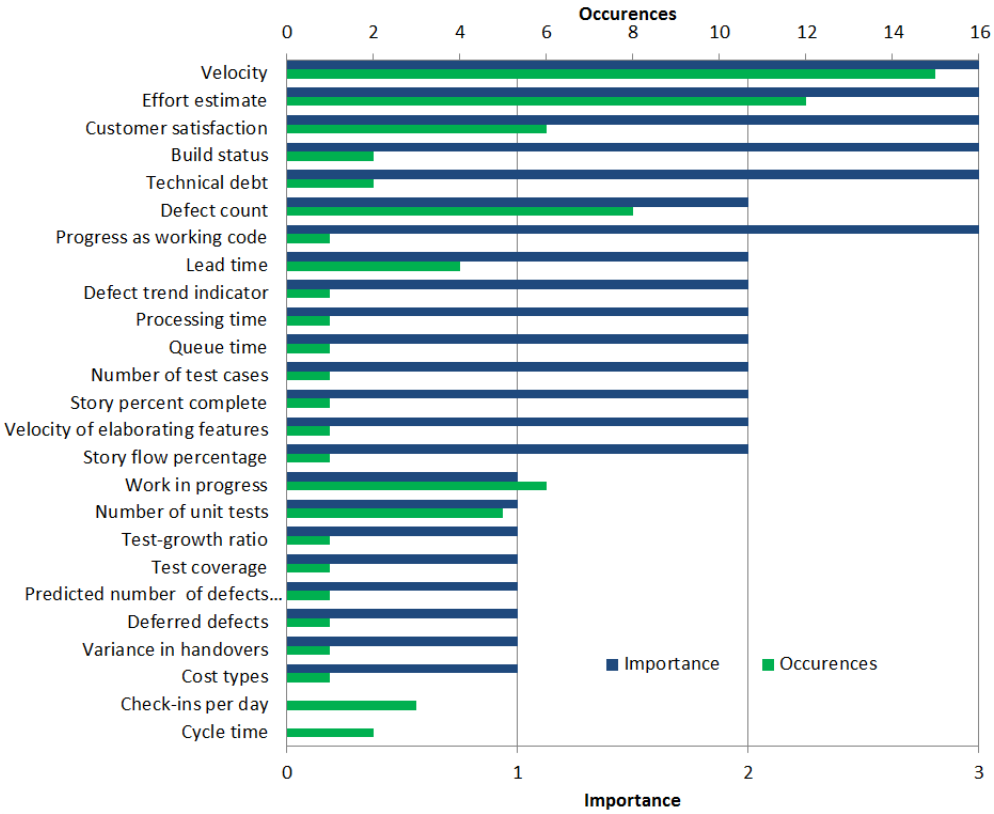

Figure 4: High influence metrics based on number of occurrences and perceived importance factor

trend indicator were seen important to decision making, and their use continued after the pilot period [S26]. The key attributes of the metrics were sufficient accuracy and ease of use.

The following metrics were considered very useful in an Agile context: number of unit tests, test coverage, test-growth ratio, and build status [S14]. The benefit of the number of unit tests was not well described except that it provided *"first insights"*. Test coverage provided information on how well the code was tested. Test-growth ratio was useful in projects where an old codebase was used as a basis for new features. Finally, fixing broken builds prevented defects from reaching customers.

4.5.2. Quantitative approach—Frequency of metrics occurrences

Velocity and effort estimate metrics were the most described metrics, with 15 and 12 occurrences. Additionally, work in progress metric occurred 6 times, and lead time was mentioned in 4 sources. Thus, following project progress and ensuring its smoothness were highly important in our sources. Furthermore,

Table 10: High influence metrics based on number of occurrences and perceived importance factor

Metric	Number of occurrences	Importance factor	Sum of ranks / 2
Velocity [S1, S2, S3, S5, S6, S8, S8, S10, S13, S16, S16, S16, S23, S27, S28]	15	3	1
Effort estimate [S3, S7, S8, S8, S9, S12, S15, S15, S15, S15, S17, S29]	12	3	1.5
Customer satisfaction [S1, S3, S7, S17, S19, S20]	6	3	2.5
Defect count [S1, S3, S5, S7 ,S7, S10, S25, S27]	8	2	5
Technical debt [S4, S4]	2	3	5
Build status [S4, S14]	2	3	5
Progress as working code [S30]	1	3	6.5
Lead time [S18, S19, S22, S24]	4	2	7
Story flow percentage [S13]	1	2	9.5
Velocity of elaborating features [S13]	1	2	9.5
Story percent complete [S29]	1	2	9.5
Number of test cases [S1]	1	2	9.5
Queue time [S18]	1	2	9.5
Processing time [S18]	1	2	9.5
Defect trend indicator [S25]	1	2	9.5
Work in progress [S17, S20, S21, S22, S23, S24]	6	1	10
Number of unit tests [S1, S5, S14, S27, S28]	5	1	11
Cost types [S21]	1	1	14
Variance in handovers [S21]	1	1	14
Deferred defects [S7]	1	1	14
Predicted number of defects in backlog [S25]	1	1	14
Test coverage [S14]	1	1	14
Test-growth ratio [S14]	1	1	14
Check-ins per day [S5, S27, S28]	3	NA	16
Cycle time [S17, S23]	2	NA	16.5

our sources also revealed that tracking quality was important. Defect count, customer satisfaction, number of unit tests, and technical debt occurred 8, 6, 5, and 2 times, respectively.

5. Discussion

This chapter discusses findings based on the results. The findings are compared with existing knowledge, and further implications are explored. Finally, the limitations of the study are discussed.

5.1. Focus of metrics in Agile development

Research Question 1: *What metrics are used in industrial Lean and Agile software development?*

Based on the results of this study, in Agile development, the targets of measurement are the product and the process, but not the people. This implies that measuring resources is not important or can be detrimental to performance, as discussed in section 5.2. One explanation is that Agile development assumes a capable team, the members of which can improve themselves without metrics. Boehm and Turner [5] acknowledge that Agile projects require people with higher methodological skills than Plan-driven projects. Also, Agile methods are more suitable to smaller products and teams, while Plan-driven methods are better for larger products and teams [5]. Based on the results of this study and prior work, we hypothesise that measuring people becomes more important when the product and team are large.

Another observation from the metric categorisation in Table 7 is that documentation, such as design specifications, is not measured. Instead, the focus is on the actual product and features, which aligns with the first Agile principle: *"Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."* [3].

Based on the metric categorisation by Fenton and Pfleeger [10] in Table 7, the following processes are measured the most: implementation, testing, and the whole development cycle. Requirements engineering, specification and design are seldom measured in Agile development beyond the measures included in 'the whole development cycle'. One possibility is that the aforementioned processes are not considered important in Agile development. Another possibility is that they are completely tied within "implementation" or "the whole development cycle" so that there is no need for separate measures. According to this study, industrial Agile teams use metrics suggested by Agile literature, but they also use custom metrics. Based on the metric comparison in Table 8, it seems that Agile teams are mostly using the metrics suggested by Agile literature. Although Kanban and LeanSD metrics were not extensively used, but that is likely due to low number of cases using Kanban or LeanSD. There were also many metrics that were not suggested in Agile literature. This implies that industrial Agile teams tend to add custom metrics on top of the metrics suggested by Agile literature.

5.2. On the Effects of and Reasons for Metric Use

Research Question 2: *What are the reasons for and effects of using metrics in industrial Lean and Agile software development?*

The categories for the reasons and effects of using metrics in Agile software development are Sprint and Project Planning, Sprint and Project Progress Tracking, Understanding and Improving Quality, Fixing Software Process Problems, and Motivating People.

When we look into individual metrics, we can see that many metrics are used for many different purposes. For example, metrics of defects are used in all categories. Even in Sprint and Project Planning, defect metrics are used as companies wish to predict the number of future defects in resource planning [S25]. Of course, some metrics are more prominent in certain phases. For example, Effort Estimate is used in planning and tracking, but not in others. Overall, based on this study it is difficult to give precise instruction of what metrics should be used for what purpose. Rather, the value of this research lies in improving understanding what practitioners typically try to achieve with metrics in Agile software development, see Table 9.

Next, we compare the effects and reasons found by this study to the reasons and effects found by other researchers, see Table 11.

Table 11: Comparison of reasons and effects of using metrics to prior research

This paper	Prior research
Planning	Project estimation [13], Improved project planning [28]
Progress tracking	Progress monitoring [13], Improved project communication [28], Management of projects [28]
Understand and improve quality	Evaluation of work products [13], Measurement is necessary for quality control and assurance [39]
Identify and fix process problems	Process improvement through failure analysis [13], Cost-effective improvement programmes [28], Process improvement [14]
Motivate people	Measurement changes behavior [11, 14]
Not found	Experimental validation of best practices [13], Alignment of software development to business objectives [28], Measurement is important for prediction, We want to predict products and processes at the stages of the software life-cycle [39], The software development process should be designed by measurable objectives, which leads to a precise definition of software quality attributes [39]

According to Jones [17], software productivity and quality are measured accurately by the most successful software companies. In this study, there was a great deal of evidence for the use of velocity, which could be seen as a measure of

productivity. Sutherland et al. [35] define velocity as a measure of productivity, but point out that it does not give a very accurate picture, since velocity does not capture business value very well. Quality was measured with defect counts and customer satisfaction metrics.

Jones [17] argues that successful software companies plan and estimate software projects accurately. Based on the results of this study , see section 4.4.1, there was a lot of emphasis on the planning and estimation of software projects. The equivalence between the results of this study and Jones' implies that industrial Agile teams are doing the right things to be successful.

In literature, there are also reasons for using metrics that we did not find in this study, see Table 11. First, *"Experimental validation of best practices"* [13] means using metrics to decide if a practice is worth using or not. This type of reasons was not in the scope of this study. This study was more focused to find metrics that would bring immediate benefits for the team. Second, *"The software development process should be designed by measurable objectives, which leads to a precise definition of software quality attributes."* [39]. Instead of defining precise quality attributes, Agile teams tend to measure the end product quality with customer based metrics (section 4.4.3) rather than rely on Traditional quality models, such as ISO/IEC 25010 [15]. Third, *"Measurement is important for prediction. We want to predict products and processes at the stages of the software life-cycle."* [39]. In this study, prediction was mostly focused to predicting post-release quality by using pre-release quality metrics, see section 4.4.3. We hypothesise that prediction is less used in Agile software development due to the uncertainty of development. Furthermore, accurate predictions would be very hard to achieve.

Regarding the effects of metric use in this study, a quote from Jones [17] can be analysed: *"The goal of applied software measurement is to give software managers and professionals a set of useful, tangible data points for sizing, estimating, managing, and controlling software projects with rigor and precision"*. Those statements are then mapped to the found effects of metric use. *"Sizing and estimating"* could be seen as actions about "Planning", *"Managing and controlling"* could be seen as actions on "Progress tracking".

This study shows that the use of metrics can motivate people and change the way people behave. Based on the results in section 4.4.5, metrics can have an effect on fixing time various issues such as defects, builds, and static analysis violations. Additionally, metrics helped people focus on reducing technical debt, e.g. implementing automatic deployment systems, increasing the number of unit tests, and preventing cherry picking of low priority but more interesting features. Naturally, it should be pointed out that metrics alone did not change the individuals' behaviour in these cases. Rather, the change in behaviour was due to paying attention to certain issues, e.g. build fixing time, and then using a metric as part of the process of increasing attention. Prior work by Goldratt [11] summarises the effect metrics can have on people nicely: "Tell me how you measure me, and I will tell you how I will behave".

This study shows that the use of metrics can have negative effects and drive dysfunctional behaviour, see section 4.4.5. Based on this, we hypothesise that

Agile methods do not provide any special protection from the dysfunctional use of metrics even when using the core metrics of Agile development, e.g., velocity [S6]. However, there was not a lot of evidence for this, although one case showed strong evidence and another hinted at negative effects of metric use [S14]. Yet it is presumable that dysfunctional use of metrics would rarely be reported, as there is a publication bias of reporting only positive results. In prior work, Hartmann and Dymond [14] discuss similar experiences of improper metrics that waste resources and skew team behaviour in counter-productive ways. Similarly, Grady [12] has experienced problems with metrics and people, so he has written a software metrics etiquette, advising among other things, against measuring individuals and against using metrics against the people who are reporting the data.

Finally, based on the results of this study, industrial Agile teams use situative metrics. Situative metrics are created and used based on a need, a solution to a problem. At a company called Systematic, they had issues with the long fix times of broken builds. They started measuring fix time of broken build and showed the time next to the coffee machine. It provoked discussion on the reasons for long fix times, and eventually the developers fixed the builds faster [S13]. Similarly, they had issues with preparations for sprints. They started measuring *percentage of stories prepared for sprint*, supported by a checklist. At Petrobras, they had problems with customers related to rework and delays. They started measuring technical debt with a technical debt board that visualised the state of different technical debt categories in their projects. This helped create awareness and address various technical debt issues [S4]. In prior work Hartmann and Dymond [14], identified short-term context driven 'diagnostics'. These diagnostics seem to be the same as the situative metrics described in this study. Based on the results of this study and prior work, we hypothesise that shor-term context driven diagnostics or situative metrics could be more unique to Agile development.

5.3. High influence metrics

Research Question 3: *What metrics have high influence in industrial Lean and Agile software development?*

5.3.1. Characteristics of high influence metrics

In this study, we identified the high influence metrics by analysing qualitatively the perceived importance of the reported metrics and quantitatively the number of occurrences, see section 4.5.2. We identified some common characteristics of high influence metrics that deserve to be pointed out. First, ease of use and ability to utilise existing tools were identified as the aspects of metrics that were perceived to be important. Second, based on the effects of metric use, it seems that the ability to provoke discussion is a characteristic of important metrics. Value Stream Maps and number of bounce backs initiated root cause analysis meetings [S3, S17]. Moreover, metrics were analysed in a reflection meeting where a problem and an improvement were identified [S28]. Third, the

ability to provide visibility of problems was perceived a useful characteristic. Technical debt board provided visibility on technical debt issues and helped create discussion to decrease technical debt [S4]. Hartmann and Dymond [14] also list as one of their Agile metric heuristics that metrics should provide fuel for meaningful conversation.

5.3.2. Frequently occurring metrics

In this study, we identified that the most often occurring metrics in literature were velocity, effort estimate, defect count, and customer satisfaction. This result suggests that industrial Agile teams value planning (effort estimate), progress tracking (velocity), pre-release quality (defect count), and post-release quality (customer satisfaction).

It is important to note that the results of this study bring forward two quality metrics among the top four high influence metrics in industrial Agile development. As shown in Table 10, the defect counts and customer satisfaction are commonly reported metrics, even though they are not directly recommended by the well known Agile or Lean methods. This result supports our earlier finding that Agile methods lack direct quality metrics, which was identified as a potential shortcoming of the methods [16]. This study reveals that such metrics as defect counts and customer satisfaction are commonly implemented by industry, and thus, perceived to be important.

Hartmann and Dymond [14] emphasise that value creation should be the primary measure of progress, which was also seen in this study [S30]. Hartmann and Dymond [14] also propose having one key metric to measure business value, preferably in agreement with the business unit. They give examples for the key metric: Return of Investment (ROI), Net Present Value (NPV), and Internal Rate of Return (IRR). However, those were not seen in this study. One reason for the absence of the aforementioned metrics in this study could be the focus of this study to the metrics of software teams instead of the metrics of the whole organisation. Furthermore, Hartmann and Dymond [14] do not provide any specific Agile metrics but rather describe how Agile metrics should be chosen and how they should be introduced to the organisation.

5.4. Mapping metric use to Agile principles

To evaluate the agility of the found metrics and their use, the results are mapped to the principles of Agile software development [3] categorised by Patel et al. [24], see Table 12. For each paragraph the categorisation by Patel et al. is used.

Communication and Collaboration was reflected by metrics providing a basis for discussion. Value Stream Maps and number of bounce backs initiated root cause analysis meetings [S3,S17]. Moreover, metrics were analysed in a reflection meeting where a problem and an improvement were identified [S28]. Furthermore, technical debt board provided visibility of technical debt issues and helped to start discussion to decrease technical debt [S4].

Team involvement was reflected in metrics that motivated teams to act and improve, see section 4.4.5. Also, to promote sustainable development, metrics

Table 12: Agile principles and software metrics. Numbers in the parenthesis() refer to the agile principles [3]

Theme [24]	Agile Principles [3]	Findings
Communication and Collaboration	Business people and developers must work together daily throughout the project (4). The most efficient and effective method of, conveying information to and within a development,team is face-to-face conversation (6).	Metrics provided a basis for discussion and increased visibility.
Team involvement	Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done (5). Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely (8).	Metrics motivated people to act, promoted sustainability development, and trust.
Reflection	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly (12).	Metrics helped to generate improvement ideas and spot problems.
Frequent delivery of working software	Our highest priority is to satisfy the customer through early and continuous delivery of valuable software (1). Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale (3). Working software is the primary measure of progress (7).	Some cases measured progress with working software but in some task completion was measured instead. Working software was measured with customer satisfaction, feedback, and customer defect reports.
Managing Changing Requirements	Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage(2).	Metrics often helped in Sprint planning
Design	Continuous attention to technical excellence and good design enhances agility (9). Simplicity–the art of maximising the amount of work not done–is essential (10). The best architectures, requirements, and designs emerge from self-organising teams (11).	Measures of technical debt, build status, violations of statistical code analysis, and for enforcing test first development promoted design quality.

were targeted to balance the flow of work, see section 4.4.2. Furthermore, people were not measured (Table 7), which indicates trust.

Reflection was directly visible in metrics that were used to identify problems and generate ideas for improvement, see section 4.4.4.

Frequent delivery of working software was directly identified in one of the studies, where the team measured progress by demonstrating the product to the customer [S30]. Additionally, there were cases where, e.g., completed webpages [S12] were the primary progress measure. Also, many metrics focused on progress tracking and timely completion of project goals, see section 4.4.2. However, some other measures from section 4.4.2 show that instead of working code Agile teams followed completed tasks and velocity metrics.

An integral part of the concept of working software is measuring post-release quality, see section 4.4.3. This was measured by customer satisfaction, feedback, and customer defect reports. It was also common to use pre-release data to predict post-release quality. Agile developers tend to measure end product quality with customer based metrics instead of with Traditional quality models, such as ISO/IEC 25010 [15].

Managing Changing Requirements was seen in the metrics that support prioritisation of features, see section 4.4.1 This allowed the rapid development of features important to the customer's business at a given time. Also, metrics like technical debt board provided a better codebase for further development.

Design was directly seen in focus for measuring technical debt, static analysis violations, and using metrics to enforce writing tests before actual code, see section 4.4.3. Additionally, the status of the build was continuously monitored, see section 4.4.3. However, the use of velocity metric had a negative effect on technical quality, see section 4.4.5. Many metrics focused on making sure that the right features were selected for implementation, see section 4.4.1, thus avoiding unnecessary work. Moreover, metrics were used to identify waste (processes where no value is added to the product), see section 4.4.4.

There were also metrics, or their use, that were not Agile in nature, such as maintaining velocity by cutting corners in quality instead of dropping features from that iteration [S6]. Also, adding people to a project to reach a certain date [S5, S17] does not seem that Agile compared to removing tasks. Furthermore, Brook's law suggests "adding manpower to a late software project makes it later" due to the lack of knowledge and training time required for new people. Moreover, the use of number of defects to delay a release [S10] is against Agile thinking, as one should rather decrease scope to avoid such a situation. Furthermore, developers at Avaya used effort estimates to predict the iteration where a feature would be completed [S29], which contradicts the idea of completing a feature within an iteration.

Some Agile metrics that work well for an Agile team, such as tracking progress by automated tests [S28] or measuring the status of the build [S14], can turn against the Agile principles if used as an external controlling mechanism. The fifth Agile principle requires trust in the team, but if the metrics are enforced outside of the team, e.g., from upper management, there is a risk that the metrics turn into control mechanisms and the benefits for the team itself

suffer.

5.5. Limitations

The large shares of specific application domains in the primary documents are a threat to external validity. Seven out of 30 studies were from the enterprise information systems domain, and especially strong was also the share of ten telecom industry studies, out of which eight were from the same company, Ericsson. Also, Israeli Air Force was the case organisation in three studies. Thus, there is a chance that the results of this study only represent the situation in particular companies. Another threat to external validity is the chosen research method, SLR. There is a great deal of industrial metric use in Agile teams that is not reported in scientific literature. So choosing another research method, e.g., a survey targeted at companies practicing Agile methods, could have produced different results. We chose to do an SLR instead of a survey, as we thought it would be better to do the SLR first and then continue with the survey.

The threats to the reliability of this research mainly include issues related to the reliability of primary study selection and data extraction. The main threat to reliability was having a first author performing the study selection and data extraction. This threat was mitigated by analysing the reliability of both study selection and data extraction as described in section 3. Additionally, the first author was supported by the second and third author in daily/weekly meetings where problematic cases regarding data extraction and study selection were discussed. Nevertheless, it is possible that researcher bias could have had an effect on the results.

Due to iterative nature of the coding process, it was challenging to make sure that all previously coded primary documents would get the same treatment whenever new codes were discovered. In addition, the researchers' coding 'sense' developed over time, so it is possible that data extraction accuracy improved in the course of the analysis. These risks were mitigated by conducting a pilot study to improve the coding scheme, get familiar with the research method, and refine the method and tools.

Some data are not explained in much detail in the primary studies, which could have caused incorrect interpretations. For example, sometimes it was hard to understand which metrics an author was referring to when a "why" was described. Moreover, we sometimes had to assume that when author described the reasons for using a tool, he would actually be talking about the metrics the tool shows.

Deciding which Agile method was used in the cases was difficult. On the other hand, it is quite natural that cases use many aspects from multiple Agile methods.

Finally, this study could have been improved by studying the reference list of the primary studies as suggested in the EBSE guidelines by Kitchenham and Charters [21]. By performing full scale snowballing to the primary studies would have increased the reliability of the findings. However in this study, we chose the database search approach, as we thought we would be unlikely to find studies

that explored precisely our research questions. The data showed that we were, right as we extracted the information of using metrics mainly from case studies of Agile software development.

6. Conclusions

This study provides researchers and practitioners an overview of the use of software metrics in industrial Agile context. This study makes three contributions. First, this study categorises the metrics found in empirical Agile studies and compares the found metrics with the metrics suggested by Agile literature. The results show that Agile teams use many metrics suggested by the Agile literature. In particular, Velocity and Effort estimate are highly popular in industrial Agile teams. However, Agile teams also use many metrics (40/102) not suggested in Agile literature. In the future, software engineering researchers should focus on popular industrial metrics if they wish to support industrial software development. Another possibility for future work is to study technical debt or build breaks, as those metrics were highly popular in the primary studies even though they were not suggested by the Agile literature.

Second, this study sheds light into the reasons for and effects of using metrics in Agile software development. The use of metrics is done in the following areas: Sprint and Project Planning, Sprint and Project Progress Tracking, Understanding and Improving Quality, Fixing Software Process Problems, and Motivating People. We think these areas show that the reasons for using metrics are similar in both the Plan-driven and Agile world. Software engineers want to plan and track their projects, they also care about the quality, they want to improve their processes, and they need to influence the team they are working with. Any of the topics identified as reasons for using metrics can be a fruitful area for future research.

Third, this study identifies high influence metrics based on the number of occurrences and statements found in the primary studies. The number of occurrences showed that Velocity, Effort estimate, and Defect count were the most popular metrics. The qualitative analysis of metric importance showed that Customer satisfaction, Technical debt, Build status, and Progress as working code as highly important metrics. Focusing research efforts on metrics that have a relatively low number of occurrences but are seen as important in the primary studies is a good choice for future research, as the industrial adaption of those metrics is still low though they can possibly have a strong influence.

Finally, this research also discovered what we here name as the "Ericsson bias". We found that 22% of our cases came from Ericsson, which is a large international telecom company. In total 28% of our cases were from telecom domain, which creates a risk for bias. Ericsson is a well-known research partner for empirical software engineering researchers around the globe, but software engineering researchers are suggested to partner also with other companies and domains to mitigate this problem in the future.

Acknowledgment

This work has been partially funded by EU FP7 Grant 318082 - UQASAR (http://www.uqasar.eu/). The authors thank the individuals of the UQASAR project and the participants and reviewers of WeTSOM2014 workshop, who provided comments on the earlier version of this paper.

References

[1] D. J. Anderson. *Kanban*. Blue Hole Press, 2010.

[2] K. Beck and C. Andres. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.

[3] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. Manifesto for agile software development, 2007.

[4] Carlo Gabriel Porto Bellini, Rita De CASsia De Faria Pereira, and JoAO Luiz Becker. Measurement in software engineering: From the roadmap to the crossroads. *International Journal of Software Engineering and Knowledge Engineering*, 18(01):37–64, 2008.

[5] B. Boehm and R. Turner. Using risk to balance agile and plan-driven methods. *Computer*, 36(6):57–66, 2003.

[6] Raymond PL Buse and Thomas Zimmermann. Information needs for software development analytics. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 987–996. IEEE Press, 2012.

[7] C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346–7354, 2009.

[8] D. S. Cruzes and T. Dybå. Recommended steps for thematic synthesis in software engineering. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, pages 275–284, 2011.

[9] T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9):833 – 859, 2008.

[10] N. E. Fenton and S. L. Pfleeger. *Software metrics: a rigorous and practical approach*. PWS Publishing Co., 1998.

[11] E.M. Goldratt. *The Haystack Syndrome: Sifting Information Out of the Data Ocean*. North River Press Publishing Corporation, 2006.

[12] R. B. Grady. *Practical software metrics for project management and process improvement*. Prentice-Hall, Inc., 1992.

[13] R. B. Grady. Successfully applying software metrics. *Computer*, 27(9): 18–25, Sept 1994.

[14] D. Hartmann and R. Dymond. Appropriate agile measurement: using metrics and diagnostics to deliver business value. In *Agile Conference, 2006*, pages 6 pp.–134, July 2006.

[15] ISO/IEC. *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. Number ISO/IEC 25010:2011. ISO/IEC, 2011.

[16] J. Itkonen, K. Rautiainen, and C. Lassenius. Towards understanding quality assurance in agile software development. In *Proceedings of the International Conference on Agility*, pages 201–207, 2005.

[17] C. Jones. *Applied software measurement: global analysis of productivity and quality*, volume 3. Mcgraw-hill New York, 2008.

[18] B. Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33:2004, 2004.

[19] B. Kitchenham. What's up with software metrics? - a preliminary mapping study. *Journal of Systems and Software*, 83(1):37–51, January 2010.

[20] B. Kitchenham and P. Brereton. A systematic review of systematic review process research in software engineering. *Information and Software Technology*, 55(12):2049–2075, 2013.

[21] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. Technical report, EBSE Technical Report EBSE-2007-01, 2007. URL `https://www.cs.auckland.ac.nz/~norsaremah/2007%20Guidelines%20for%20performing%20SLR%20in%20SE%20v2.3.pdf`.

[22] E. Kupiainen, M. V. Mäntylä, and J. Itkonen. Why are industrial agile teams using metrics and how do they use them? In *Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics*, pages 23–29. ACM, 2014.

[23] J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977.

[24] C. Patel, M. Lycett, R. Macredie, and S. de Cesare. Perceptions of agility and collaboration in software development practice. In *System Sciences, 2006. HICSS '06. Proceedings of the 39th Annual Hawaii International Conference on*, volume 1, pages 10c–10c, 2006.

[25] K. Petersen. Is lean agile and agile lean? In Ali H. Dogru and Veli Bicer, editors, *Modern Software Engineering Concepts and Practices: Advanced Approaches*. Hershey.

[26] Kai Petersen and Claes Wohlin. Context in industrial software engineering research. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 401–404. IEEE Computer Society, 2009.

[27] M. Poppendieck and T. Poppendieck. *Lean software development: An agile toolkit*. Addison-Wesley Professional, 2003.

[28] K. Pulford, A. Kuntzmann-Combelles, and S. Shirlaw. *A quantitative approach to software management: the AMI handbook*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[29] S. Purao and V. Vaishnavi. Product metrics for object-oriented systems. *ACM Computing Surveys (CSUR)*, 35(2):191–221, 2003.

[30] Danijel Radjenović, Marjan Heričko, Richard Torkar, and Aleš Živkovič. Software fault prediction metrics: A systematic literature review. *Information and Software Technology*, 55(8):1397–1418, 2013.

[31] P. Rodríguez, J. Markkula, M. Oivo, and K. Turula. Survey on agile and lean usage in finnish software industry. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '12, pages 139–148, New York, NY, USA, 2012. ACM.

[32] K. Schwaber and M. Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002.

[33] K. Schwaber and J. Sutherland. The scrum guide. *Scrum. org, July*, 2013.

[34] G. A. F. Seber. *The Estimation of Animal Abundance and Related Parameters*. Blackburn Press, 2002.

[35] J. Sutherland, G. Schoonheim, and M. Rijk. Fully distributed scrum: Replicating local productivity and quality with offshore teams. In *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, pages 1–8, Jan 2009.

[36] CMMI Product Team. CMMI for development, version 1.3. Technical Report CMU/SEI-2010-TR-033, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 2010. URL `http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=9661`.

[37] VersionOne. 7th annual state of agile survey. http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf, 2012.

[38] J. P. Womack, D.T. Jones, and D. Roos. *The machine that changed the world: The story of lean production.* Simon and Schuster, 2007.

[39] H. Zuse. *A Framework of Software Measurement.* Walter de Gruyter, 1998.

Primary studies

[S1] N. Abbas, A. M. Gravell, and G. B. Wills. The impact of organization, project and governance variables on software quality and project success. In *Proceedings - 2010 Agile Conference, AGILE 2010*, pages 77–86, Orlando, FL, 2010

[S2] D. J. Anderson. Stretching agile to fit cmmi level 3-the story of creating msf for cmmi® process improvement at microsoft corporation. In *Agile Conference, 2005. Proceedings*, pages 193–201. IEEE, 2005

[S3] T. H. Cheng, S. Jansen, and M. Remmers. Controlling and monitoring agile software development in three dutch product software companies. In *Proceedings of the 2009 ICSE Workshop on Software Development Governance, SDG 2009*, pages 29–35, Vancouver, BC, 2009

[S4] P. S. M. dos Santos, A. Varella, C. Ribeiro Dantas, and D. Borges. Visualizing and managing technical debt in agile development: An experience report. In *Agile Processes in Software Engineering and Extreme Programming*, volume 149 of *Lecture Notes in Business Information Processing*, pages 121–134, 2013

[S5] Y. Dubinsky, D. Talby, O. Hazzan, and A. Keren. Agile metrics at the israeli air force. In *Proceedings - AGILE Confernce 2005*, volume 2005, pages 12–19, Denver, CO, 2005

[S6] A. Elssamadisy and G. Schalliol. Recognizing and responding to "bad smells" in extreme programming. In *Proceedings - International Conference on Software Engineering*, pages 617–622, Orlando, FL, 2002

[S7] P. Green. Measuring the impact of scrum on product development at adobe systems. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, Koloa, Kauai, HI, 2011

[S8] D. R. Greening. Enterprise scrum: Scaling scrum to the executive level. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, Koloa, Kauai, HI, 2010

[S9] N. C. Haugen. An empirical study of using planning poker for user story estimation. In *Proceedings - AGILE Conference, 2006*, volume 2006, pages 23–31, Minneapolis, MN, 2006

[S10] P. Hodgetts. Refactoring the development process: Experiences with the incremental adoption of agile practices. In *Proceedings of the Agile Development Conference, ADC 2004*, pages 106–113, Salt Lake City, UT, 2004

[S11] P. Hodgkins and L. Hohmann. Agile program management: Lessons learned from the verisign managed security services team. In *Proceedings - AGILE 2007*, pages 194–199, Washington, DC, 2007

[S12] N. Hong, J. Yoo, and S. Cha. Customization of scrum methodology for outsourced e-commerce projects. In *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, pages 310–315, Sydney, NSW, 2010

[S13] C. R. Jakobsen and T. Poppendieck. Lean as a scrum troubleshooter. In *Proceedings - 2011 Agile Conference, Agile 2011*, pages 168–174, Salt Lake City, UT, 2011

[S14] A. Janus, R. Dumke, A. Schmietendorf, and J. Jager. The 3c approach for agile quality assurance. In *Emerging Trends in Software Metrics (WETSoM), 2012 3rd International Workshop on*, pages 9–13, 2012

[S15] S. Keaveney and K. Conboy. Cost estimation in agile development projects. In *Proceedings of the 14th European Conference on Information Systems, ECIS 2006*, Goteborg, 2006

[S16] V. Mahnic and N. Zabkar. Measuring progress of scrum-based software projects. *Electronics and Electrical Engineering*, 18(8):73–76, 2012

[S17] P. Middleton, P. S. Taylor, A. Flaxel, and 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(4):387–403, 2007

[S18] S. Mujtaba, R. Feldt, and K. Petersen. Waste and lead time reduction in a software product customization process with value stream maps. In *Proceedings of the Australian Software Engineering Conference, ASWEC*, pages 139–148, Auckland, 2010

[S19] K. Petersen and C. Wohlin. The effect of moving from a plan-driven to an incremental software development approach with agile practices: An industrial case study. *Empirical Software Engineering*, 15(6):654–693, 2010

[S20] K. Petersen and C. Wohlin. Software process improvement through the lean measurement (spi-leam) method. *Journal of Systems and Software*, 83(7): 1275–1287, 2010

[S21] K. Petersen and C. Wohlin. Measuring the flow in lean software development. *Software - Practice and Experience*, 41(9):975–996, 2011

[S22] K. Petersen. A palette of lean indicators to detect waste in software maintenance: A case study. *Lecture Notes in Business Information Processing*, 111 LNBIP:108–122, 2012

[S23] R. Polk. Agile & kanban in coordination. In *Proceedings - 2011 Agile Conference, Agile 2011*, pages 263–268, Salt Lake City, UT, 2011

[S24] M. Seikola, H. M. Loisa, and A. Jagos. Kanban implementation in a telecom product maintenance. In *Proceedings - 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011*, pages 321–329, Oulu, 2011

[S25] M. Staron, W. Meding, and B. Söderqvist. A method for forecasting defect backlog in large streamline software development projects and its industrial evaluation. *Information and Software Technology*, 52(10):1069–1079, 2010

[S26] M. Staron and W. Meding. Monitoring bottlenecks in agile and lean software development projects - a method and its industrial use. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6759 LNCS:3–16, 2011

[S27] D. Talby, O. Hazzan, Y. Dubinsky, and A. Keren. Reflections on reflection in agile software development. In *Proceedings - AGILE Conference, 2006*, volume 2006, pages 100–110, Minneapolis, MN, 2006

[S28] D. Talby and Y. Dubinsky. Governance of an agile software project. In *Proceedings of the 2009 ICSE Workshop on Software Development Governance, SDG 2009*, pages 40–45, Vancouver, BC, 2009

[S29] V. Trapa and S. Rao. T3 - tool for monitoring agile development. In *Proceedings - AGILE Conference, 2006*, volume 2006, pages 243–248, Minneapolis, MN, 2006

[S30] J. Trimble and C. Webster. From traditional, to lean, to agile development: Finding the optimal software engineering cycle. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, pages 4826–4833, Wailea, Maui, HI, 2013

Appendix A. Search strings

The database searches were performed in following three consecutive searches. The reason for the three consecutive searches was that we found that the first search, which was limited to "computer science" subject area did not include certain central conference proceedings. We extended the same key word search to cover more generic "engineering" subject area in the second search. The third search was another extension of the same search make sure that potentially relevant papers classified under the "business" subject area were included.

The first search (September 19th, 2013):

TITLE-ABS-KEY(software AND (agile OR lean OR "crystal method" OR "crystal clear" OR dsdm OR "dynamic systems development method" OR fdd OR "feature driven development" OR "agile unified process" OR "agile modeling" OR scrumban OR kanban OR scrum OR "extreme programming" OR xp) AND (measur* OR metric OR diagnostic OR monitor*)) AND (LIMIT-TO(SUBJAREA, "COMP")) AND (LIMIT-TO(LANGUAGE, "English"))

resulted to 512 hits.

The second search (November 7th, 2013):

TITLE-ABS-KEY(software AND (agile OR lean OR "crystal method" OR "crystal clear" OR dsdm OR "dynamic systems development method" OR fdd OR "feature driven development" OR "agile unified process" OR "agile modeling" OR scrumban OR kanban OR scrum OR "extreme programming" OR xp) AND (measur* OR metric OR diagnostic OR monitor*)) AND (LIMIT-TO(LANGUAGE, "English")) AND (LIMIT-TO(SUBJAREA, "ENGI")) AND (EXCLUDE (SUBJAREA, "COMP") OR EXCLUDE(SUBJAREA, "PHYS") OR EXCLUDE(SUBJAREA,"MATE") OR EXCLUDE (SUBJAREA, "BUSI") OR EXCLUDE(SUBJAREA, "MATH") OR EXCLUDE(SUBJAREA, "ENVI") OR EXCLUDE (SUBJAREA, "EART") OR EXCLUDE(SUBJAREA, "DECI") OREXCLUDE (SUBJAREA, "ENER"))

resulted to 220 hits.

The third search (December 10th, 2013):

TITLE-ABS-KEY(software AND (agile OR lean OR "crystal method" OR "crystal clear" OR dsdm OR "dynamic systems development method" OR fdd OR "feature driven development" OR "agile unified process" OR "agile modeling" OR scrumban OR kanban OR scrum OR "extreme programming" OR xp) AND (measur* OR metric OR diagnostic OR monitor*)) AND (LIMIT-TO(LANGUAGE, "English")) AND (LIMIT-TO(SUBJAREA, "BUSI")) AND (EXCLUDE (SUBJAREA, "ENGI") OR EXCLUDE(SUBJAREA, "COMP"))

resulted to 42 hits.

Appendix B. Inclusion and exclusion criteria

Inclusion criteria

- Papers that present the use and experiences of metrics in an agile industry setting.

Exclusion criteria

- Papers that do not contain empirical data from industry cases.
- Papers that are not in English.
- Papers that do not have agile context. There is evidence of clearly non-agile practices or there is no agile method named. For example, paper mentions agile but case company has only three releases per year.
- Paper is only about one agile practice, which is not related to measuring.
- Papers that do not seem to have any data about metric usage. Similarly, if there are only a few descriptions of metrics but no other info regarding reasons or usage.
- Papers that have serious issues with grammar or vocabulary and therefore it takes considerable effort to understand sentences.
- Papers where the setting is not clear or results cannot be separated by setting, for example surveys where there is data both from academia and industry.
- Papers where the metrics are only used for the research. For example, author measures which agile practices correlate with success.

Appendix C. Quality assessment questions

Based on the quality evaluation form by Dybå and Dingsøyr [9].

1. Is this a research paper?
2. Is there are a clear statement of the aims of the research?
3. Is there an adequate description of the context in which the research was carried out?
4. Was the research design appropriate to address the aims of the research?
5. Was the recruitment strategy appropriate to the aims of the research?
6. Was there a control group with which to compare treatments?
7. Was the data collected in a way that addressed the research issue?
8. Was the data analysis sufficiently rigorous?
9. Has the relationship between researcher and participants been considered adequately?
10. Is there a clear statement of findings?
11. Is the study of value for research or practice?

Appendix D. Definions of metrics

Table D.13: Definitions of found metrics

Primary study	Metric	Definition
[S10]	# of defects	Issues found from quality assurance cycle including differences from expected behavior.
[S7]	# of defects found in system test	Number of defects found in system test phase.
[S25]	# of defects in backlog	All known and unresolved defects in the project.
[S7]	# of open defects	Number of open defects on the current release per day.
[S22]	# of requirements per phase	Number of requirements (work items/features) per phase.
[S14]	# of unit tests	Number of unit tests.
[S23]	Average velocity	Not clearly defined in primary study.
[S4, S14]	Build status	Build broken or not.
[S5, S27, S28]	Burndown	Remaining human resource days versus the remaining work days.
[S7]	Burndown	Not defined in primary study.
[S1]	Business value delivered	Not defined in primary study. Probably means delivered features per timeframe.

[S19]	Change requests per requirement	Amount of change requests from customer per requirement.
[S5, S27, S28]	Check-ins per day	Number of commits (code, automated test, specification) per day.
[S17]	Common tempo time	Net working days available per number of (work) units required.
[S12]	Completed web pages	Completed web pages.
[S16]	Cost performance index	Not defined in primary study.
[S3]	Critical defects sent by customer	No detailed definition in primary study.
[S1]	Customer satisfaction	Not defined in primary study.
[S17]	Customer satisfaction (Kano analysis)	Not clearly defined in primary study.
[S17]	Cycle time	Not defined in primary study.
[S23]	Cycle time	Time it takes for x size story to be completed.
[S1]	Defect count after testing	Not defined in primary study. Probably means amount of defects after first round of testing.
[S25]	Defect trend indicator	Indicates if amount of defects in the coming week will increase, stay the same or decrease from this week.
[S7]	Defects deferred	Not defined in primary study. Probably means the amount of defects that are known but are not fixed for the release.
[S9]	Effort estimate	Estimated effort per story in ideal pair days.
[S12]	Effort estimate	Not clearly defined in primary study.
[S15, S15, S15, S15]	Effort estimate	Not defined in primary study.
[S17]	Effort estimate kits	Tasks are broken down into kits of two to five staff-days of work.
[S19]	Fault slips	Amount of issues that should have been found already in the previous phase.
[S5]	Faults per iteration	Faults per iteration.
[S13]	Fix time of failed build	Fix time of failed build.
[S19]	Implemented vs wasted requirements	Ratio of implemented requirements and wasted requirements. Not all requirements are always implemented but some work is put into them, e.g., in the form of technical specification.
[S20]	Inventory of requirements over time	Amount of requirements (features/work items) in specific work phase over time.
[S18]	Lead time	The average time it takes for one request to go through the entire process from start to finish.
[S19, S22]	Lead time	Time it takes for requirement to go through a subprocess or the whole process.
[S24]	Lead time	Not clearly defined in primary study.
[S19]	Maintenance effort	Costs related to fixing issues that have been found and reported by customers.

[S7]	Net Promoter Score	Not defined in primary study. Probably measures how likely customers will recommend the product to another customer.
[S5, S27, S28]	Number of automated passing test steps	Number of automated passing test steps.
[S17]	Number of bounce backs	Not defined in primary study. Probably the amount of defects that should have not occurred anymore if a root cause would have been fixed earlier.
[S27]	Number of new and open defects	Number of new and open defects.
[S20]	Number of requests from customers	Not defined in primary study.
[S1]	Number of test cases	Not defined in primary study.
[S3]	Open defects	Not defined in primary study.
[S8]	Operations' velocity	Not defined in primary study. Probably Operations department's completed story points per time unit.
[S13]	Percentage of stories prepared for sprint	Percentage of stories prepared for sprint.
[S16]	Planned velocity	Not clearly defined in primary study.
[S25]	predicted # of defects	Predicted number of defects in backlog in the coming week.
[S18]	Processing time	The time the request is being worked on by one person or a team.
[S30]	Progress as working code	Product is demonstrated to the customer who then gives feedback.
[S23]	Pseudo velocity	Not clearly defined in primary study.
[S26]	Queue	Number of units remaining to be developed/processed by a given phase or activity.
[S18]	Queue time	The average time between sub-processes that the request sits around waiting.
[S21]	Rate of requirements per phase	Rate of requirements flow from a phase to next phase.
[S16]	Release burndown	Amount of work remaining till the release.
[S3]	Remaining task effort	Not defined in primary study.
[S21]	Requirement's cost types	Cost distribution of a requirement.
[S11]	Revenue per customer	Amount of revenue from customer per feature.
[S1]	Running tested features	Not defined in primary study. Probably means amount of features delivered to customer that are passing unit tests.
[S16]	Schedule performance index	Not defined in primary study.
[S16]	Sprint burndown	Amount of work remaining till the end of sprint.
[S29]	Story complete percentage	Not clearly defined in primary study.
[S29]	Story estimate	Estimated days to complete the story.
[S6]	Story estimates	Estimated time to develop a story.

[S13]	Story flow percentage	Estimated implemention time per actual implementation time * 100.
[S7]	Story points	Not defined in primary study.
[S8]	Story points	Estimated effort to complete the story in programmer days.
[S12]	Task done	Task done.
[S8]	Task effort	Estimated effort to complete the task in programmer hours.
[S12]	Task's expected end date	Date when a task is estimated to be finished.
[S3]	Team effectiveness	Not defined in primary study.
[S4]	Technical debt board	Shows the status of each technical debt category per team.
[S4]	Technical debt in effort	Technical debt in amount of hours it would take to fix all the issues increasing technical debt calculated by third party tool called Sonar.
[S14]	Test coverage	How much Source Code executed during Test Execution.
[S3]	Test failure rate	Not defined in primary study.
[S14]	Test growth ratio	Difference of amount of tests per difference of amount of Source Code.
[S3]	Test success rate	Not defined in primary study.
[S26]	Throughput	Number of units processed by a given phase or activity per time.
[S21]	Variance in handovers	Changes in amount of handed over requirements.
[S2]	Velocity	Amount of developed scenarios per developer per week.
[S6]	Velocity	Not defined in primary study.
[S8]	Velocity	Not defined in primary study.
[S10]	Velocity	Feature points developed per iteration.
[S13]	Velocity of elaborating features	Not clearly defined in primary study. Probably the time it takes to clarify a feature from customer into requirements that can be implemented.
[S13]	Velocity of implementing features	Not clearly defined in primary study. Probably the time it takes to implement a feature.
[S14]	Violations of static code analysis	Amount of violations to static code analysis rules from tools like Findbugs, PMD and Checkstyle.
[S17]	Work in progress	Amount of features or feature level integrations team is working on.
[S23]	Work in progress	Amount of stories per work phase.
[S24]	Work in progress	Amount of work items per phase.