

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Eetu Kupiainen

Reasons and Effects of Metric Use in Industrial Agile Teams

A Systematic Literature Review

Master's Thesis
Espoo, April 7, 2014

Supervisor: Prof. Casper Lassenius
Instructors: Juha Itkonen D.Sc. (Tech.)
Mika Mäntylä D.Sc. (Tech.)

Aalto University
 School of Science
 Degree Programme in Computer Science and Engineering

ABSTRACT OF
 MASTER'S THESIS

Author:	Eetu Kupiainen		
Title:	Reasons and Effects of Metric Use in Industrial Agile Teams A Systematic Literature Review		
Date:	April 7, 2014	Pages:	73
Major:	Software Engineering	Code:	T3003
Supervisor:	Prof. Casper Lassenius		
Instructors:	Juha Itkonen D.Sc. (Tech.) Mika Mäntylä D.Sc. (Tech.)		
<p>Agile development methods are increasing in popularity, yet there are limited studies on the reasons and effects of metrics use in industrial agile development.</p> <p>This paper presents results from a systematic literature review. Based on the study, the results indicate that the reasons for the use metrics are focused on the following areas: Planning, Progress tracking, Understand and improve quality, and Identify problems. Similarly, the effects of metric use are focused on the following areas: Planning actions, Reactive actions, Motivate people and Create improvement ideas. The findings are mapped against agile principles and it seems that the use of metrics supports the principles with some deviations.</p> <p>Furthermore, this study identifies important metrics based on statements and amount of evidence. Also, the study shows a lot of evidence on the use of planning and tracking metrics. Finally, the dysfunctional use of metrics and the negative effects of metric use can be an interesting future research topic.</p>			
Keywords:	agile, metrics, measurement, systematic, literature, review		
Language:	English		

Aalto-yliopisto
 Perustieteiden korkeakoulu
 Tietotekniikan koulutusohjelma

DIPLOMITYÖN
 TIIVISTELMÄ

Tekijä:	Eetu Kupiainen		
Työn nimi:	Mittareiden käytön syyt ja seuraukset ketterissä tiimeissä teollisuudessa Systemaattinen kirjallisuuskatsaus		
Päiväys:	7. huhtikuuta 2014	Sivumäärä:	73
Pääaine:	Ohjelmistotuotanto ja liiketoiminta	Koodi:	T3003
Valvoja:	Prof. Casper Lassenius		
Ohjaajat:	Tkt Juha Itkonen Tkt Mika Mäntylä		
<p>Ketterät ohjelmistokehitysmenetelmät ovat lisänneet suosiotaan, mutta tutkimuksia, joissa tutkittaisiin syitä mittareiden käytölle ja niiden vaikutuksille, on varsin rajallisesti teollisen ketterän kehityksen osalta.</p> <p>Tämä tutkimus esittää tuloksia systemaattisesta kirjallisuuskatsauksesta. Perustuen tähän tutkimukseen, tulokset esittävät, että syyt mittarien käytölle keskittyy seuraaviin alueisiin: Suunnittelu, Etenemisen seuranta, Laadun ymmärtäminen ja parantaminen, ja Ongelmien tunnistaminen. Lisäksi, mittareiden vaikutukset keskittyvät seuraaviin alueisiin: Suunnittelutoimet, reaktiiviset toimet, Ihmisten motivointi ja Parannusideoiden luominen. Löydöksiä verrataan ketterän kehityksen periaatteisiin ja näyttää siltä, että mittareiden käyttö tukee ketteriä periaatteita lukuuottamatta muutamia poikkeuksia.</p> <p>Tämä tutkimus myös listaa tärkeitä mittareita, jotka tunnistettiin lausumien ja esiintymismäärien avulla. Lisäksi, tämä tutkimus osoittaa, että suunnittelu- ja seurantamittareiden käytöstä on paljon näyttöä. Lopuksi, mittareiden käytön negatiiviset vaikutukset voi olla mielenkiintoinen tulevaisuuden tutkimusalue.</p>			
Asiasanat:	ketterä, kehitys, mittarit, mittaus, systemaattinen, kirjallisuuskatsaus		
Kieli:	Englanti		

Acknowledgements

I want to thank my instructors D.Sc. Juha Itkonen and D.Sc. Mika Mäntylä for their invaluable feedback, tangible guidance and tireless attitude towards my questions. Also, I want to thank B.Sc Kim Dikert and M.Sc. Timo Lehtinen for relaxing and thoughtful discussions. Furthermore, I would like to express my thanks to L.Sc Kristian Rautiainen and D.Sc Jari Vanhanen for all their help. Finally, I would like to thank my supervisor Prof. Casper Lassenius for bringing up improvement ideas in this thesis. This work has been funded by EU FP7 Grant 318082 (U-QASAR, <http://www.uqasar.eu/>).

Espoo, April 7, 2014

Eetu Kupiainen

Abbreviations and Acronyms

KPI	Key Performance Indicator
LeanSD	Lean Software Development
SLR	Systematic literature review
WIP	Work in progress
XP	Extreme Programming

Contents

Abbreviations and Acronyms	5
1 Introduction	8
1.1 Structure of the thesis	9
2 Background	10
2.1 Agile software development	10
2.2 Software measurement	12
2.3 Systematic Literature Review	13
3 Review method	14
3.1 Protocol development	14
3.2 Search and selection process	14
3.3 Data extraction	15
3.4 Data synthesis	17
4 Results	19
4.1 Overview of studies	19
4.2 Quality evaluation of the primary studies	22
4.3 Metrics	24
4.4 Why are metrics used?	29
4.4.1 Planning	29
4.4.2 Progress tracking	30
4.4.2.1 Project progress	30
4.4.2.2 Simplify complexity and increase visibility . .	31
4.4.2.3 Accomplishing project goals	31
4.4.2.4 Balance workflow	32
4.4.3 Understand and improve quality	32
4.4.3.1 Understand level of quality	33
4.4.3.2 Increase quality	33
4.4.3.3 Ensure level of testing	33

4.4.4	Identify problems	34
4.5	What are the effects of metric use?	36
4.5.1	Planning actions	36
4.5.2	Reactive actions	37
4.5.3	Motivate people	38
4.5.4	Create improvement ideas	40
4.6	Important metrics	42
4.6.1	Important metrics in terms of statements	42
4.6.2	List of important metrics	44
5	Discussion	46
5.1	Focus of metrics in agile development	46
5.2	Findings on effects and reasons of metric use	47
5.2.1	Negative effects of metric use	47
5.2.2	Situative metrics	48
5.2.3	Reasons for metric use	48
5.3	Important metrics	51
5.3.1	Characteristics of important metrics	51
5.3.2	Important metrics	51
5.4	Mapping metric use to agile principles	52
5.5	Limitations	53
6	Conclusions	55
	References	56
	Primary studies	61
A	Search strings	65
B	Inclusion and exclusion criteria	67
C	Quality assesment questions	68
D	Definions of metrics	69

Chapter 1

Introduction

Software metrics have been studied for decades and several literature reviews have been published. Yet, the literature reviews have been written from an academic viewpoint. For example, Catal and Diri (2009) review fault prediction metrics, Purao and Vaishnavi (2003) review metrics for object oriented systems and Kitchenham (2010) performs a mapping of most cited software metrics papers. According to the researcher's knowledge there are no systematic literature reviews on the actual use of software metrics in the industry.

Agile software development is becoming increasingly popular in the software industry. The agile approach seems to be contradicting with the traditional metrics approaches. For example, the agile emphasizes working software over measuring progress in terms of intermediate products or documentation, and embracing the change invalidates the traditional approach of tracking progress against a pre-made plan. However, at the same time agile software development highlights some measures that should be used, e.g., burndown graphs and 100% automated unit testing coverage. However, metric research in the context of agile methods remains scarce.

The goal of this study is to review the literature of actual use of software metrics in the context of agile software development. This study lays out the current state of metrics usage in industrial agile software development based on literature. Moreover, the study uncovers the reasons for metric usage as well as highlights the effects of metric use. Furthermore, this study is part of an EU funded project U-QASAR¹. The goal in the U-QASAR project is to develop an analysis tool for software development projects with heavy focus on metrics. This study aims to support the tool development by providing understanding about metric usage in industrial agile software development,

¹<http://www.uqasar.eu/>

and by providing suggestions for metrics used in the tool. This study covers the following research questions:

- **Research Question 1:** *What metrics are used in industrial agile software development?*
- **Research Question 1.1:** *What are the entities measured in industrial agile software development?*
- **Research Question 1.2:** *How do the metrics found in industrial agile software development compare with those suggested by agile literature?*
- **Research Question 2:** *Why are metrics used in industrial agile software development?*
- **Research Question 3:** *What are the effects of metric use in industrial agile software development?*
- **Research Question 4:** *What metrics are important in industrial agile software development?*

1.1 Structure of the thesis

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

Chapter 2

Background

This chapter describes the key concepts used in this study. First, section 2.1 describes agile software development. Second, section 2.2 describes the benefits of software measurement. Third, section 2.3 describes the used research method, Systematic Literature Review.

2.1 Agile software development

Agile development methods have emerged to the software world ruled by traditional heavyweight methods. 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.

Agile manifesto created by agile enthusiasts (Beck et al., 2007) lists four core agile values:

- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*
- *Customer collaboration over contract negotiation*
- *Responding to change over following a plan*

Furthermore, popular agile development methods include Scrum (Schwaber and Beedle, 2002), Extreme Programming (Beck and Andres, 2004), Lean Software Development (Poppendieck and Poppendieck, 2003) and Kanban (Anderson, 2010). Scrum (Schwaber and Beedle, 2002) got its name from rugby, where the team groups up and plans its next move. Similarly in software development, the scrum team has daily meetings where they see what has been done and what should be done next. On 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 the daily meeting where the status of the tasks is tracked. At the end of the sprint, a sprint review is organized where all the tasks are reviewed. Learning is emphasized every sprint with a sprint retrospective meeting where issues related to work practices are discussed and improvements are suggested.

Extreme Programming (XP) (Beck and Andres, 2004) got its name from a set of principles and practices that were emphasized to the extreme. For example, if you think testing is a good practice then test all the time with automated unit tests. Similarly, if you think code reviews are beneficial, then do constant code reviews by programming in pairs. Additionally, XP aims at embracing change. This is achieved in XP by continuously refactoring the code base, creating and maintaining a comprehensive unit test suite and designing simple. Changes in business requirements can then be flexibly developed. Communication is also very important in XP. Communication is handled with unit tests, pair-programming and having a customer available to answer business questions.

Lean Software Development (LeanSD) (Poppendieck and Poppendieck, 2003) contains many attributes from Scrum and XP, but it also utilises principles from Lean manufacturing, which Toyota used to build their competitive edge Womack et al. (2007). 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*

The aforementioned agile methods are often well defined and they require their practices to be used comprehensively. Kanban (Anderson, 2010) on the other hand is more of an evolutionary process model that allows each Kanban implementation to be different, suited for each context. Some principles, however, are the same in Kanban. 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 according to demand. The pull system is enabled by limiting work in progress (WIP). For example, if the WIP limit is set to two for development and there are already two work items under development then no more work can be started before either of them is completed.

Many different agile software development methods conforming to similar principles comprise agile software development. The agile methods emphasise short development cycles, frequent face-to-face communication and continuous learning. In this Systematic Literature Review, only studies that are of agile software development are considered.

2.2 Software measurement

“If you can not measure it, you can not improve it...” said Lord Kelvin, a mathematical physicist and engineer. Similarly, according to Fenton and Pfleeger (1998), we use metrics every day to understand, control and improve what we do and how we do it. According to Jones (2008), who is a founder of Software Productivity Research with a knowledge base of thousands of software projects, the top performing software companies such as IBM and Microsoft extensively use metrics in their business, while the lower performing teams do not. Furthermore, measurement is a key technology for successful software development and maintenance. Pulford et al. (1995) describe the following benefits of metric use:

- *Improved project planning and management of projects.*
- *Alignment of software development to business objectives.*
- *Cost-effective improvement programmes.*
- *Improved project communication.*

Similarly, Grady (1992) lists reasons for metric use for practitioners and managers:

- *A basis for estimates.*
- *To track project progress.*
- *To determine (relative) complexity.*

- *To help us to understand when we have achieved a desired state of quality.*
- *To analyze defects,*
- *and to experimentally validate best practices.*
- *In short: they help us to make better decisions.*

There are many benefits to software metrics, for example being able to improve many aspects of software projects, and in general, make better decisions. This study tries to collect software metrics and understand the reasons and effects for metric use. Additionally, this study tries to distinguish important metrics.

2.3 Systematic Literature Review

Systematic Literature Review (SLR) is a research method originated from the field of medicine (Kitchenham, 2004). There are three main reasons for conducting an SLR (Keele, 2007). First, to aggregate and synthesise existing knowledge regarding a research topic. Second, identify gaps in earlier research. Third, provide background information to start investigating a new research topic. Moreover, SLR provides a repeatable research method, a method that when applied properly, should provide the exactly same results irrespective of the person who is performing it. Furthermore, the detailed documentation of steps in SLR allows in-depth evaluation of the rigorousness of the conducted study. However, SLR requires considerably more effort than traditional reviews. Usually, SLRs are conducted by researchers and PhD students.

SLR is an audible and trustworthy research method for multiple purposes. In this study, SLR is used to understand why metrics are used and what are the effects of metric use in agile software development. Furthermore, SLR is used to collect the used metrics, and distinguish important metrics.

Chapter 3

Review method

Systematic literature review (SLR) was chosen as a research method because the study is more about trying to understand a problem instead of trying to find a solution to it. Also, there was already existing literature that could be synthesized.

3.1 Protocol development

A guide for SLRs by Kitchenham (2004) was used as a basis for developing the review protocol. Additionally, an SLR on agile development (Dybå and Dingsøyr, 2008) and an SLR on SLR (Kitchenham and Brereton, 2013) were used to further understand the challenges and opportunities of SLRs. The protocol was also iterated in weekly meetings with the instructors, as well as in a pilot study.

3.2 Search and selection process

The strategy for finding primary studies was following:

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

Table 3.1 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

¹<http://www.scopus.com>

Table 3.1: Paper selection funnel

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

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 inclusion criteria: *papers that present empirical findings on the industrial use and experiences of metrics in agile context*. The papers were excluded based on multiple criteria, mainly due to not conforming to requirements regarding empirical findings and agile and industrial context. 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 XP Conference 2013 was searched manually because it could not be found through Scopus.

In stage 2, papers were included and excluded by the researcher based on their title and abstract. As the quality of abstracts can be poor in computer science (Kitchenham, 2004), full texts were also skimmed through in case of unclear abstracts. Unclear cases were discussed with the instructors 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 instructor. The level of agreement was 'substantial' with Kappa 0.67 (Landis and Koch, 1977).

Stage 3 included multiple activities in one workflow. Selection by full text was done, data was coded and quality assessment was done. Once again, if there were unclear papers, they were discussed in the meetings. Also, selection of 7 papers was conducted by the second instructor with an 'almost perfect' agreement, Kappa 1.0 (Landis and Koch, 1977).

3.3 Data extraction

Integrated coding was selected for data extraction strategy (Cruzes and Dyba, 2011). Integrated coding includes having a start list of codes as well as

creating new codes if necessary (ground-up). It provided focus for research questions but flexibility regarding findings. Deductive coding would have been too restraining 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 related to the metric
- Context

The coding started with the researcher reading the full text and marking interesting quotes with a temporary code. After, reading the full text the researcher checked each quote and coded again with an appropriate code based on the built understanding. In weekly meetings with the instructors, a rule set for collecting metrics was slowly built:

- 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 Visual QDA (Qualitative Data Analysis), version 7.1.x was used to collect and synthesize the qualitative data. The amount of found quotes per code can be seen in Table 3.2. To evaluate the repeatability of finding the same metrics, second instructor coded metrics from three primary studies. Capture-recapture method (Seber, 2002) was then used which showed that 90% of metrics were found.

A quality assessment form adopted from (Dybå and Dingsøy, 2008) 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 scale for the relevancy factor is:

Table 3.2: 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

- 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

3.4 Data synthesis

Metrics were grouped based on similarity to enable categorization in Table 4.5 and Table 4.9. 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 Dyba (2011). The process started by going through all quotes within one code and giving each quote a more descriptive code describing the quote at a high level. Then the descriptive codes were organized in groups based on their similarity, see e.g., Figure 3.1 These groups were then given high level codes which are seen as categories in Table 4.7 and Table 4.8.

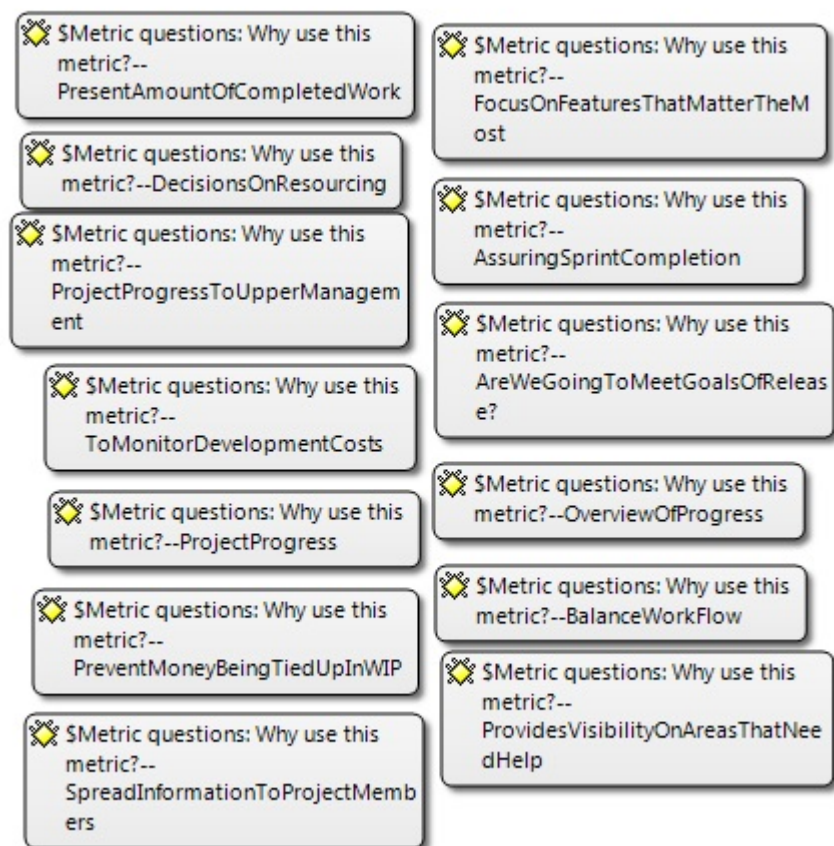


Figure 3.1: Reason for metric use category named 'Progress tracking' was formed by organizing descriptive codes into a group based on their similarity

Chapter 4

Results

This chapter 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), categorizes them based on the entity that is measured (RQ1.1) and compares them with metrics suggested by agile literature (RQ1.2). Section 4.4 describes the reasons for using metrics (RQ2). Section 4.5 describes the effects of metric use (RQ3). Section 4.6 describes important metrics (RQ4) by statements from the primary studies as well as by the amount of evidence from primary studies.

4.1 Overview of studies

This section gives an overview of the primary studies. Table 4.1 shows the distribution of primary studies by publication channel. Table 4.2 lists the primary studies by context factors.

The study identified 30 primary studies. Primary studies were published in 12 different journals, conferences or workshops, see Table 4.1. A large share of primary studies (43%) was published in Agile Conference. Rest of the studies were published in a wide range of journals, conferences and workshops.

Primary studies and their context information can be seen in Table 4.2. 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 with 18 studies, then experience report with 8 studies, multi-case with 3 studies and finally survey with 2 studies.

Agile method for the studies was identified based on the assessment of

Table 4.1: The publication distribution of 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

the researcher. A specific method was chosen if it seemed to be a primary method in the case. If it was unclear what agile method was used, then 'NA' was set as the agile method. Based on the results, Scrum was the most used agile method (35%) in primary studies. XP was the second most used agile method (20%) while LeanSD and Kanban were used in 5% of the cases. In 33% of the cases, the used agile method was unclear.

Telecom was the most represented domain with 10 cases, enterprise information systems was the second with 7 cases and web applications with 4 cases. There were 15 cases that were other domains or cases without domain information.

Table 4.2: Overview of primary studies

ID	Year	Resear. meth.	Agile method	Team size	Domain
[S1]	2010	Survey	NA	NA	NA
[S2]	2005	Experience r.	MSF v4.0 ¹	NA	NA
[S3]	2009	Multi-case	NA/Scrum/ Scrum	2-10/2-7/4- 8	ERP/Graphic design plug- in/Facility man- agement
[S4]	2013	Experience r.	Scrum	25 teams	Software for oil and gas industry
[S5]	2005	Single-case	XP	15	Enterprise infor- mation system
[S6]	2002	Experience r.	XP	50	Enterprise re- source 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 so- lutions/Criminal justice system development
[S11]	2007	Single-case	Scrum	500	Security services

¹Microsoft Solutions Framework v4.0

[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
[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 with a scale from 0 to 1. The evaluation form was adopted from (Dybå and Dingsøy, 2008). The detailed list of quality evaluation questions can be found in appendix C.

Table 4.3: Quality evaluation of primary studies

Study	Res- earch	Aim	Con-R.d- text esign	Sam- pling	Ctr.l. grp	Data coll.	Data anal.	Re- flex.	Find- ings	Val- ue	Tot- al	Rele- vancy	
[S1]	1	1	1	1	1	0	1	1	1	1	10	2	
[S2]	0	0	0	0	1	0	0	0	0	1	2	2	
[S3]	1	1	0	1	0	0	0	0	0	0	3	2	
[S4]	0	0	0	0	1	0	0	0	0	1	2	3	
[S5]	1	1	1	1	1	0	1	1	0	1	9	3	
[S6]	0	0	1	0	1	0	0	0	0	1	3	2	
[S7]	0	0	0	0	0	1	1	1	0	1	5	2	
[S8]	0	0	0	0	0	1	0	0	0	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	1	3	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		

Additionally, a relevancy factor was assigned for each study describing its relevancy for 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 value for research, context and findings scored the highest. There were 13 primary studies with a total score of 8, 9 or 10, and 11 primary studies with a total score of 1, 2 or 3.

4.3 Metrics

This section lists, categorizes and compares the found metrics from the SLR. First, the found metrics are listed by primary study in Table 4.4 answering **Research Question 1: *What metrics are used in industrial agile software development?***. Second, the metrics are categorized according to the categorization by Fenton and Pfleeger (1998) in Table 4.5, answering **Research Question 1.1: *What are the entities measured in industrial agile software development?***. Third, the found metrics are compared with the metrics suggested by agile methods in Table 4.6, answering **Research Question 1.2: *How do the metrics found in industrial agile software development compare with those suggested by agile literature?***. Metrics were only collected if their *reason of use*, *effect of use* or *importance* was described. A total of 102 metrics were found in the primary studies. Definitions of metrics can be found from appendix D.

Table 4.4: RQ1: Metrics by primary studies

ID	Metrics
[S1]	Business value delivered, customer satisfaction, defect count after testing, number of test cases, running tested features
[S2]	Velocity
[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 / effort estimate / effort estimate / 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

According to the categorization in Table 4.5, metrics were used to measure products, test plans, code, builds, features, requirements and defects. Most of the entities in Products class were measured internally, except the products entity, which was measured mostly externally. Furthermore, test-

Table 4.5: RQ1.1: Metric categorization based on Fenton and Pfleeger (1998)

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, 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 [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]	

ing, implementation and the whole development cycle were measured mostly internally in the Processes class. Only two metrics are related to measuring Resources class.

Table 4.6: RQ1.2: Comparison of metrics found in agile literature compared to the metrics found in this study

Metrics suggested by literature	Method	Scrum	XP	Kanban	LeanSD	NA
Effort estimate	Scrum, XP, LeanSD	2[S7] ² , 1[S8], 2[S8], 3[S12]	1[S9]			1[S15], 2[S15], 3[S15], 4[S15], 6[S17], 1[S29]
Velocity ³	Scrum, XP, LeanSD	3[S8], 2[S10], 4[S8], 1[S16], 2[S16], 5[S16], 1[S23]	1[S6], 1[S5], 1[S27], 1[S28]	4[S23]	5[S13]	1[S2], 5[S3]
Written and passed unit tests	XP, LeanSD		3[S5], 3[S27], 2[S28], 5[S14]			5[S1]
Actual development time	XP					
Load factor	XP					
Work in progress	Kanban	1[S21]		2[S23], 2[S24]		4[S17], 2[S20], 1[S22]
Lead time	Kanban		5[S19]	1[S24]	1[S18]	2[S22]
Due date performance	Kanban					
Throughput	Kanban					1[S26]
Issues and blocked work items	Kanban					
Flow efficiency	Kanban					

²Number prior to study reference is the index of the metric in Table 4.4

³Includes total work remaining from Scrum and effort left from Scrum and XP.

Initial quality	Kanban, LeanSD	3[S7], 4[S7]	4[S5], 1[S10], 4[S27]	3[S1], 2[S3], 2[S25]
Failure load	Kanban			2[S17]
Cycle time	LeanSD		3[S23]	3[S17]
Value Stream Maps (Work time, wait time)	LeanSD			2[S18], 3[S18]
Amount of writ- ten and passed ac- ceptance tests per iteration	LeanSD			4[S1], 3[S3], 4[S3]
Not suggested		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]	2[S5], 2[S27], 3[S28], 1[S14], 2[S14], 3[S14], 4[S14], 1[S19], 2[S19], 3[S19], 4[S19]	1[S13], 2[S13], 3[S13], 4[S13] 1[S1], 2[S1], 1[S3], 6[S3], 1[S17], 5[S17], 1[S20], 1[S25], 3[S25], 2[S26], 2[S29], 1[S30]

Metrics found in the primary studies are compared with the metrics suggested by agile literature in Table 4.6. The metrics found in literature come from Scrum (Schwaber and Sutherland, 2013), XP (Beck and Andres, 2004), Kanban (Anderson, 2010) and LeanSD (Poppendieck and Poppendieck, 2003). Four rightmost columns in Table 4.6 are describing the agile method used in the primary studies. In some primary studies it was hard to identify a specific agile method, thus 'NA' is used to describe those cases. The number before primary study reference defines the index of the metric in Table 4.4.

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 primary studies. Many metrics (40 of 102, 39%) found in the primary studies were not suggested in agile literature.

4.4 Why are metrics used?

Research Question 2: *Why are metrics used in industrial agile software development?*

The following sections describe the reasons for the use of metrics. Table 4.7 lists the categories for reasons of metric use by sources. Some of the descriptions for the reasons of metric use might seem to be incomplete. This is because the effects of metric use are described in the following section 4.5. Also, some reason categories might seem to be describing more the effect than the reason, e.g., Balance workflow in section 4.4.2.4. However, the reasons for metric use are described here for Balance workflow and then the actual actions and effects are described in the next section 4.5.

Table 4.7: Reasons for the use of metrics by sources

Categories	Sources
Planning	[S6, S8, S9, S11, S12, S16, S17, S21, S23, S24, S25, S29]
Progress tracking	[S2, S4, S5, S7, S8, S12, S13, S16, S17, S20, S21, S22, S23, S25, S27, S28]
Understand and improve quality	[S3, S4, S5, S7, S14, S17, S19, S22, S28, S29]
Identify problems	[S2, S13, S16, S18, S20, S21, S22, S25, S29]

4.4.1 Planning

Prioritization of tasks was one of the main activities metrics were used for. At Objectnet, effort estimates were used to prioritize the features for the next release and as basis for resourcing [S9]. Teams at Adobe Systems used effort estimates to prioritize activities based on relative value and relative effort [S8]. At Verisign Managed Security Services, they used revenue per customer to prioritize their backlog [S11]. At Timberline Inc, they used Kano analysis as a 'voice of customer' so that prioritization decisions could be based on facts instead of political power [S17]. Practitioners at Ericsson used cost types, rate of requirements over phases and variance in handovers for short term decisions related to requirements' prioritization, staff allocation and planning decisions [S21].

Metrics were used to estimate the size and amount of features that could be taken under development. Velocity was used to improve effort estimates for next the planning session, which helped to estimate the scope for the next iteration [S16]. 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]. At Ericsson product maintenance team, lead time was used to understand if all planned corrections can be completed before release date [S24]. At Avaya Communications, they used story estimates to predict the iteration where a feature would be completed [S29].

Other planning uses for metrics were 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 a Korean e-commerce company, they marked tasks done and undone which made it possible to take undone tasks for the next iteration [S12]. Also, they marked expected end date for tasks so the next person in workflow could plan their own work as effectively as possible thus reducing idle time. 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, predicted number of defects was used to plan the removal of defects [S25]. If the removal of defects would not be well planned, it could cause delays for the release and thus increase costs for the project.

4.4.2 Progress tracking

Reasons for metrics use in progress tracking are divided into project progress, increase visibility, accomplishing project goals and balance workflow.

4.4.2.1 Project progress

Metrics were used to monitor the progress of the project. Completed web pages metric was used as a measure of progress at Korean e-commerce company [S12]. 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]. 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 story percent complete metric to give an assess-

ment of progress [S29]. However, a team at NASA Ames Research Center did not want to spend resources on estimating features and instead they focused on designing and developing their software solution [S30]. Every six weeks they demonstrated their progress to the customer with working code.

Metrics were also used to give a higher level understanding about progress. Release burndown showed project trends and could be used to predict completion date [S16]. Also, 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] and ensure good progress to external observers and ensure that key risks were under control [S5,S27,S28].

4.4.2.2 Simplify complexity and increase 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, technical debt board was used to make technical debt issues visible and easier to manage [S4]. Metrics (burndown, check-ins per day, number of automated passing test steps, number of open and new defects) were used to replace individual perception with facts [S27].

Metrics were used to keep the team informed. At Ericsson, defect trend indicator was used to monitor defect backlog and spread the information to project members [S25]. At WMS Gaming, 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.

4.4.2.3 Accomplishing project goals

Metrics were used to understand if project goals can be achieved. At Timberline Inc, there was a need for a simple indicator that would quickly tell if a project was under control [S17]. They used common tempo time to understand if project was in target for delivery. At Microsoft Corporation, they monitored work in progress to predict lead time which in turn would predict project schedule [S2]. At Adobe Systems, sprint burndown was used to tell

the team if they were on track regarding the sprint commitments [S7]. Similarly at Mamdas, burndown was used to see whether the team could meet their goals, and if not, what could be done [S5]. Burndown was also used to mitigate the risk where developers spend too much time perfecting features over finishing all tasks of the iteration [S28]. Furthermore at a Slovenian publishing company, release burndown made the correlation clear between work remaining and team's progress in reducing it [S16]. Story flow percentage was used so that a developer could finish a story in a steady flow [S13]. Story implementation flow metric describes how efficiently a developer has been able to complete a story compared with the estimate.

4.4.2.4 Balance 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, 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 notice 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 avoid people from committing only at the end of the iteration, and instead verify that work is spread evenly across iterations. At WMS Gaming, they had problems with large tasks blocking other work, so they set a rule that only certain size of tasks (8 story points) can be taken for development [S23].

4.4.3 Understand and improve quality

The following sections describe how metrics were used to understand the quality of the product both before and after release. Also, the sections will describe that metrics were used to improve the quality of the product and ensure that the product will be tested thoroughly.

4.4.3.1 Understand level of quality

Metrics were used to understand the level of quality after the release. 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 which 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].

4.4.3.2 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, they 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, violations of static code analysis metric was used to prevent the existence of 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].

4.4.3.3 Ensure level of testing

Metrics were used to make sure the product is 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 is 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 minimized [S17]. Large amount of work in progress would contain many unidentified defects which would need to be eventually discovered. At

Mamdas, using number of automated passing test steps decreased the risk that the product would be unthoroughly tested [S5]. Similarly, number of automated passing test steps was used to make sure regression tests are ran and passed every iteration. Finally, story percent complete metric supported test driven development by requiring unit tests to be written for progress tracking [S29].

4.4.4 Identify problems

Metrics were used to identify problems, bottlenecks and waste in the process. Cumulative number of work items over time metric was used to identify bottlenecks in the development process [S22]. Similarly, monitoring work in progress was used to identify blocked work items and the development phase where the blockage occurred [S2]. Cost types, rate of requirements over phases and variance in handovers were used for process improvement by spotting bottlenecks and uneven requirement flows [S21]. For example, many requirements were transferred to System Test phase, but only a small amount of requirements were transferred to Ready for Release phase, see Figure 4.1.

Metrics were able to identify waste, as in development phases where no value is added, in software processes. At Ericsson, Value Stream Maps (VSM) were used to spot waste in the development process [S18]. In another case at Ericsson, long lead times led to the identification of waste of waiting [S22]. Similarly, measuring story flow percentage allowed the identification of waste related to context shifts [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 showed if 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, where the last 20% of iteration content takes 80% of the time [S29]. With the metrics that their tool T3 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.

Metrics were also used to find improvement opportunities. Number of work items per phase and lead time was used to spot instabilities in the process [S20]. They had set control limits to the metrics and if a measured

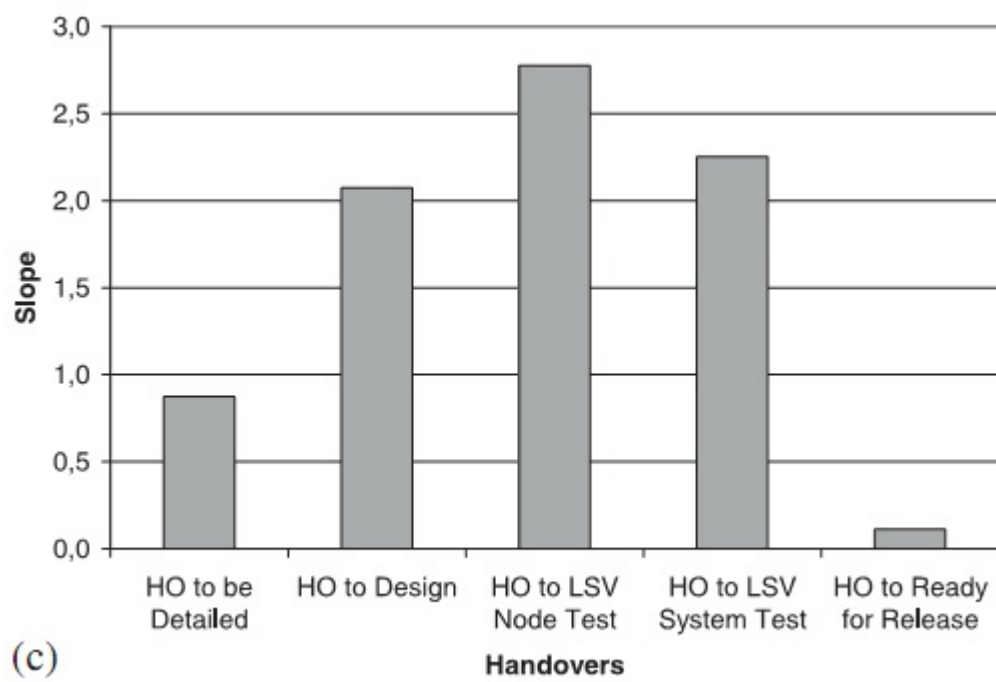


Figure 4.1: Rate of requirements identified a bottleneck in System Test phase [S21]

value was outside the control limits it meant that there was some kind of instability.

4.5 What are the effects of metric use?

Research Question 3: *What are the effects of metric use in industrial agile software development?*

The following subsections describe the effects of metric use. Table 4.8 lists the effect categories by sources. Some of the categories for the reasons and the effects of metrics are close to each other so sometimes it can be hard to understand why some categories are listed under the reasons and not under the effects. For example section 4.4.2.4 describes that the purpose of metrics was the balancing of workflow and then the actual actions and effects are described under section 4.5.

Table 4.8: Effects of metric use by sources

Categories	Sources
Planning actions	[S2, S11, S12, S23]
Reactive actions	[S3, S5, S10, S14, S16, S17, S28]
Motivate people	[S3, S4, S6, S13, S14, S17, S25]
Create improvement ideas	[S8, S10, S13, S14, S17, S18, S20, S21, S22, S26, S27, S28]

4.5.1 Planning actions

This section describes the planning actions that happened due to the use of metrics.

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 revenue per customer metric to allow higher valued features to be prioritized higher in the backlog [S11].

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. Similarly, effort estimates were used to scope the iteration and if there were tasks that could not be completed before a release date then they were excluded from the backlog [S12]. 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].

Expected date of task completion was used so that other team members could plan their own work [S12]. For example, a developer could know when she can start implementation because the designer had informed the expected date of completion for the design.

4.5.2 Reactive actions

This section describes reactive actions that occurred due to the use of metrics.

Metrics were used to cut down the scope of an iteration or to add more resources if it did not seem that all tasks could be completed with current pace. When component level burndown was used to notice that a component was behind schedule at Mamdas, resources were added and scope was reduced for the release [S5]. Similarly, release burndown showed that work remaining was not decreasing fast enough so the scope of the release was decreased [S16]. Furthermore, if common tempo time would indicate 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 imbalanced workload the work could be reorganized to achieve more balanced workflow. Similarly, if team effectiveness is 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 reduction of tasks and working overtime.

Metrics were also used to react to quality 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. At Escrow.com, number of defects was used to delay a release when too many defects were noticed in a QA cycle [S10]. At T-Systems International, quality manager interpreted the results of static code analysis from the build tool and he would then make plans for necessary refactorings [S14]. When amount of written and passed unit tests was not increasing, an alarm was raised at Mamdas [S28]. The issue was discussed in a reflection meeting where they understood that too much work was put to 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.

4.5.3 Motivate people

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

Metrics were used to motivate people to react faster to problems. 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 organization 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 on the reasons for 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 in minutes after commits, which helped to create a culture where developers react with high priority to broken builds [S4]. This helped to keep 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 the behavior of people. At Petrobras, they used a technical debt board to discuss technical debt issues in their projects, see Figure 4.2. In the meetings, team members agreed which technical debt issues they would focus in solving until the next meeting [S4]. Additionally, team members sought help from the architecture team for reducing 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 the 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 thus preventing them from working on lower priority, but more interesting, features. Finally, at Ericsson defect trend indicator created crisis awareness and motivated the developers to take actions to avoid possible problems [S25].

There can also be negative effects in using metrics. Using 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 thor-

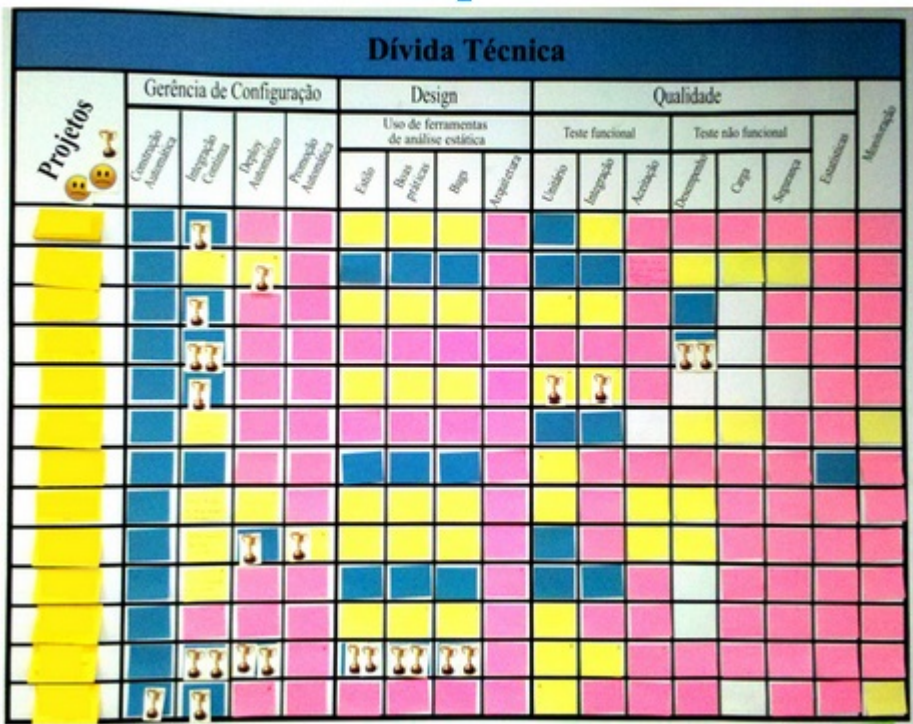


Figure 4.2: Technical debt board was used to discuss technical debt issues [S4]

oughness of the testing in hopes to maintain the velocity. Similarly, [S14] also hints towards dysfunctional use of metrics, for example developers causing a broken build if broken build is used as a KPI (Key Performance Indicator).

4.5.4 Create improvement ideas

This section describes improvement ideas that were created based on metrics.

At Ericsson, lead time, processing time and queue time metric were used to identify waste of extra process - (a requirement would wait for long time before a full specification) requirement specification [S18]. 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 if they want to pursue that requirement further. Furthermore, 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 marketing unit and development unit at least for the more complex requirements. Finally, there was a waste of waiting in the design phase which could be improved by starting real work only when the purchase order is received, not when requests are received.

Lead time, processing time and queue time metric were used to identify waste of waiting in testing phases [S18]. The improvement suggestion was to provide an earlier beta version and making testing phases parallel. Many of the improvement ideas came from meetings where Value Stream Maps (VSM) were used as a base for discussion.

Cost types, rate of requirements over phases and variance in handovers were used to identify bottlenecks at Ericsson [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 in order to accomplish the continuous delivery capabilities. Similarly at another case at Ericsson, 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].

Rate of requirements over time was used to identify problems in the development process [S20]. One improvement suggestion was to change from push to 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. Similar solution was applied at Timberline Inc. where requirements inventory 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 possible by creating hot deployment strategies and providing better documentation.

At an Israel Air force IT department, Mamdas, they were using burn-down 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 dividing work items 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 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: percentage of stories ready 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 amount 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 iteration summary meeting because it can be hard to analyze metrics without understanding the context [S27]. Similarly at Ericsson, number of work items per phase was used to ask 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 values of number of work items were outside the control limits one could discuss with the developers about the workload [S22].

At Systematic, after analyzing 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 manager

could change coding style guide and code standards based on the results of *violations to static code analysis* metric [S14].

4.6 Important metrics

Research Question 4: *What metrics are important in industrial agile software development?*

The following subsections describe first which metrics were described as important, and then second, list important metrics based on the amount of evidence and an importance factor.

4.6.1 Important metrics in terms of statements

This section describes metrics that were described as important. 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 project success in surveys, they were considered important.

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]. Also, a minimum of 60% for story flow percentage was identified as a key limit. Similarly, velocity for elaborating features should be as fast as velocity of implementing features. Also, they said that using both aforementioned metrics “*drive behaviors to let teams go twice as fast as they could before*”.

Story percent complete metric was considered valuable since it embraces test driven development - no progress is made before a test is written [S29]. Also, story percent complete metric was considered more accurate than previously used metric - however that metric was not mentioned. Moreover, story percent complete metric gave a normalized measure of progress compared to developer comments about progress. Additionally, story percent complete 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 prioritization [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.

At another case at Ericsson, Value Stream Maps (VSM) were used to visualize problem areas, and facilitate discussion for possible improvements [S18]. Practitioners valued how the maps were easy to understand. Metrics that were used to build VSM were lead time, processing time and queue time. Similarly, technical debt board, which visualized the status of technical debt, was considered important because it gave a high level understanding about the problems [S4]. Moreover, the technical debt board was then used to plan actions to remove the technical debt. Furthermore, it was proven to be useful in their context.

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 77% measured customer satisfaction often or always. Also, the more often customer satisfaction would be measured the more likely it would be that the project would have good code quality and the project would succeed. Similarly, 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 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 agile context: number of unit tests, test coverage, test-growth ratio and build status [S14]. The benefit for 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 old codebase was used as a basis for new features. Finally, fixing broken builds prevented defects reaching customers.

4.6.2 List of important metrics

Important metrics are listed in Table 4.9 sorted by the amount of evidence and an importance factor. The amount of evidence is based on the amount of cases describing the metric. Metrics that were only mentioned by name without any reasons of use, effect of use, or importance were not taken into account. The importance factor is an assessment of the metric's importance by the researcher from 1 (low) to 3 (high) based on the statements in section 4.6.1. Importance factor is not given to metrics that were not described as important. The list includes only metrics that were described as important in section 4.6.1 or metrics that were described in two or more cases.

Velocity and effort estimate metrics were the most described metrics with 15 and 12 occurrences, and they were both considered important. Defect count, customer satisfaction and work in progress metrics occurred 8, 6 and 6 times respectively. Number of unit tests, lead time and check-ins per day occurred 5, 4 and 3 times. Technical debt, build status and cycle time occurred 2 times.

Table 4.9: RQ4: Important metrics based on the amount of evidence and an importance factor

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

Chapter 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 agile software development?*

Research Question 1.1: *What are the entities measured in industrial agile software development?*

Based on the results of this study, in agile development, target for measurement is the product and the process, but not the people. Based on the metric categorization by Fenton and Pfleeger (1998) in Table 4.5, measurement is focused towards Products and Processes, but not Resources. This implies that measuring resources is not important. One implication is that agile development assumes a capable team, which can improve themselves without metrics. Boehm and Turner (2003) acknowledge that agile projects require people with higher methodological skills than plan-driven projects. Also, agile methods are more suitable to be used in smaller products and teams while plan-driven methods are better for larger products and teams boehm2003using. Based on the results of this study and prior work, the researcher hypothesises that measuring people becomes more important when the product and team is large.

Another possibility for the lack of measuring people is that measuring resources, especially people, can lead to dysfunctional behaviour, as discussed

in section 5.2.1. Another observation from the metric categorization in Table 4.5 is that documentation is not measured, e.g., design specifications. 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.*” (Beck et al., 2007).

Based on the metric categorization by Fenton and Pfleeger (1998) in Table 4.5, the following processes are measured the most: implementation, testing and the whole development cycle. This indicates that requirements engineering, specification and design are not measured in agile development. One possibility is that aforementioned processes are not considered important in agile development. Another possibility is that they are completely tied within ‘implementation’ so that they are not separated.

Research Question 1.2: *How do the metrics found in industrial agile software development compare with those suggested by agile literature?*

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 4.6, it seems that agile industry is using most of 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 Findings on effects and reasons of metric use

Research Question 2: *Why are metrics used in industrial agile software development?*

Research Question 3: *What are the effects of metric use in industrial agile software development?*

5.2.1 Negative effects of metric use

This study shows that use of metrics can have negative effects. Based on the results in section 4.5.3, metrics can be used to cause negative effects and drive

dysfunctional behaviour. Based on the results, the researcher hypothesises 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]. Hartmann and Dymond (2006) also discuss similar experiences of improper metrics that waste resources and skew team behaviour in counter-productive ways. Similarly, Grady (1992) has experienced problems with metrics and people, thus he had written a software metrics etiquette, for example advising against measuring individuals, and not using metrics against the people who are reporting the data. Furthermore, Goldratt (2006) said “Tell me how you measure me, and I will tell you how I will behave”, which highlights the complexity of measuring people.

5.2.2 Situative metrics

Based on the results of this study, industrial agile teams use situative metrics. Situative metrics are created and used based on a need, e.g., as a solution to a problem. At Systematic, they had issues with build’s long fix times. 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 with 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 to create awareness and address various technical debt issues [S4].

Hartmann and Dymond (2006) also identified short-term context driven ‘diagnostics’. ‘Diagnostics’ seem to be the same as the situative metrics described in this study. Based on the results of this study and prior work the researcher hypothesises that short term context driven ‘diagnostics’ or ‘situative metrics’ could be something more unique to agile development.

5.2.3 Reasons for metric use

The categories for reasons for metric use can be seen in Table 4.7. This answers the question why metrics are used in industrial agile development. The found reasons are compared with reasons found by others in Table 5.1.

According to Jones (2008), software productivity and quality are measured accurately by the most successful software companies. In this study, there were a great deal of evidence on the use of velocity, which could be seen as a measure of productivity. Sutherland et al. (2009) defines velocity

Table 5.1: Comparison of found reasons for metric use in this study compared with reasons found by others

Reasons for metric use from this study	Benefits / reasons for metric use from other sources
Planning	Project estimation (Grady, 1994), Improved project planning (Pulford et al., 1995)
Progress tracking	Progress monitoring (Grady, 1994), improved project communication (Pulford et al., 1995), management of projects (Pulford et al., 1995)
Understand and improve quality	Evaluation of work products (Grady, 1994), Measurement is necessary for quality control and assurance (Zuse, 1998)
Identify problems	Process improvement through failure analysis (Grady, 1994), Cost-effective improvement programmes (Pulford et al., 1995), Process improvement Hartmann and Dymond (2006)
Not found	Experimental validation of best practices (Grady, 1994), Alignment of software development to business objectives (Pulford et al., 1995), Measurement is important for prediction. We want to predict products and processes at the stages of the software life-cycle. (Zuse, 1998), The software development process should be designed by measurable objectives, which leads to a precise definition of software quality attributes. (Zuse, 1998)

as a measure of productivity, but identifies that it does not give a very accurate picture, since velocity does not capture the amount of business value very well. Quality was measured with defect count metrics and customer satisfaction metrics.

Jones (2008) argues that successful software companies plan and estimate software projects accurately. Based on the results of this study (section 4.4.1 and section 4.5.1), there was a lot of emphasis on the planning and estimation of software projects. The equivalence the results of this study to Jones' points for successful companies implies that industrial agile teams are doing the right things to be successful.

In literature, there were also reasons for metrics use that were not found in this study, see Table 5.1. First, *“Experimental validation of best practices”* (Grady, 1994) means using metrics to decide if a practice is worth using or not. This type of metrics were not in the scope 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.”* (Zuse, 1998). Instead of defining precise quality attributes, agile developers tend to measure the end product quality with customer based metrics (section 4.4.3.1) instead of the traditional quality models, such as ISO/IEC 25010 (ISO/IEC, 2010). Third, *“Measurement is important for prediction. We want to predict products and processes at the stages of the software life-cycle.”* (Zuse, 1998). In this study, prediction was mostly focused to predicting post-release quality by using pre-release quality metrics, see section 4.4.3.1. The researcher hypothesises 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 found effects of metric use in this study, a quote from Jones (2008) 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 (Table 4.8) in this study. *“Sizing and estimating”* could be seen as “Planning actions”, *“managing and controlling”* could be seen as “Reactive actions”. Researcher was not able to find similar effects of metric use as “Motivate people” would suggest. Maybe this is something more unique to agile development. Maybe the development in agile is more centered on the people, and metrics are used as a tool to drive appropriate behaviour.

5.3 Important metrics

Research Question 4: *What metrics are important in industrial agile software development?*

5.3.1 Characteristics of important metrics

This study identifies characteristics of important metric. This study describes important metrics based on descriptions of importance and the amount of evidence, see section 4.6.2. There are some characteristics of important metrics that can be pointed out: ease of use and ability to utilise existing tools. Also, based on the effects of metric use, it seems that the ability to provoke discussion is a characteristic for 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]. Furthermore, technical debt board provided visibility on technical debt issues and it helped to create discussion to decrease technical debt [S4]. Hartmann and Dymond (2006) also list as one of their agile metric heuristics that metrics should provide fuel for meaningful conversation.

5.3.2 Important metrics

This study identified important metrics: velocity, effort estimate, defect count and customer satisfaction being top 4 in terms of amount of evidence and described importance. Based on the found important metrics, the researcher argues that industrial agile teams value planning (effort estimate), progress tracking (velocity), pre-release quality (defect count) and post-release quality (customer satisfaction).

Hartmann and Dymond (2006) emphasize that creation of value should be the primary measure of progress - which was also seen in this study [S30]. Hartmann and Dymond (2006) also propose to have one key metric to measure business value, preferably agreed 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 lack of aforementioned metrics in this study could be the focus of this study to team metrics instead of metrics of the whole organization. Furthermore, Hartmann and Dymond (2006) do not provide any specific agile metrics but rather describe how agile metrics should be chosen and how they should be introduced to the organization.

Also, Hartmann and Dymond (2006) provide a set of heuristics for agile metrics.

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 Beck et al. (2007) categorized by Patel et al. (2006). For each paragraph the naming by Patel et al. is used and references to the principles of agile software development are provided by numbers.

Communication and Collaboration (4th and 6th agile principles (Beck et al., 2007)) 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 on technical debt issues and it helped to create discussion to decrease technical debt [S4].

Team involvement (5,8) was reflected in metrics that motivated team to act and improve, see section 4.5.3. Also, to promote sustainable development metrics were targeted to balance the flow of work, see section 4.4.2.4. Furthermore, people were not measured (Table 4.5), which indicates trust.

Reflection (12) was directly visible in metrics that were used to identify problems, see section 4.4.4. Furthermore, metrics helped to find improvement ideas, see section 4.5.4.

Frequent delivery of working software (1,3,7) 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 web-pages [S12] were the primary progress measure. Also, many metrics focused on progress tracking, see section 4.4.2.1, and timely completion of project goals, see section 4.4.2.3. However, some other measures from section 4.4.2.1 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 4.4.3.1. 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 the end product quality with customer based metrics instead of the traditional quality models, such as ISO/IEC 25010 (ISO/IEC, 2010).

Managing Changing Requirements (2) was seen in the metrics that support prioritization of features, see section 4.4.1 and section 4.5.1. This al-

lowed the rapid development of features important for the customer's business at a given time. Also, metrics like technical debt board provided a better codebase for further development.

Design (9,10,11) was directly seen in focus for measuring technical debt and using metrics to enforce writing tests before actual code, see section 4.4.3.3. Additionally, the status of the build was continuously monitored, see section 4.4.3.2. However, the use of velocity metric had a negative effect on technical quality, see section 4.5.3. 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, which were not agile in nature. For example, 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, adding people can have a negative impact to progress, considering the lack of knowledge and training time required. Moreover, the use of number of defects to delay a release [S10] is against agile thinking as one should rather decrease the 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 organization in three studies.

The threats to reliability in this research include mainly issues related to the reliability of primary study selection and data extraction. The main threat to reliability was having a single researcher performing the study selec-

tion and data extraction. It is possible that researcher bias could have had an effect on the results. This threat was mitigated by analysing the reliability of both study selection and data extraction as described in chapter 3. Another threat to reliability 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 to companies practicing agile methods could have produced different 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 researcher's coding 'sense' developed over time, so it is possible that data extraction accuracy improved during the analysis. In order to mitigate these risks a pilot study was conducted to improve the coding scheme, get familiar with the research method and refine the method and tools.

Some data from low scoring papers, e.g [S3], are not explained very detailed, which could have caused incorrect interpretations. Also, only the researcher conducted the quality evaluation, which could have an impact on the actual quality scores.

Deciding which agile method was used in the cases was difficult. But on the other hand it is quite natural that cases use many aspects from multiple agile methods.

Chapter 6

Conclusions

This study presents the results of a systematic literature review of 30 primary studies. According to the researcher's knowledge there are no previous systematic reviews of metric use in the context of industrial agile software development. This study makes four contributions. First, this study categorizes metrics found in empirical agile studies and compares the found metrics with the metrics suggested by agile literature. Second, this study provides descriptions of why metrics are used to support agile software development. Third, this study describes the effects metrics have for agile software development. Fourth, this study identifies important metrics based on statements and amount of evidence.

The results indicate that the reasons for the use metrics are focused on the following areas: Planning, Progress tracking, Understand and improve quality, and Identify problems. Similarly, the effects of metric use are focused on the following areas: Planning actions, Reactive actions, Motivate people and Create improvement ideas.

This study provides researchers and practitioners with an overview of the metric use in agile context and documented reasonings and effects behind the proposed metrics. This study can be used as a source of relevant studies regarding researchers' interests and contexts.

Finally, this study identified a few propositions for future research on measuring in agile software development. First, the applicable quality metrics for agile development and the relationship between pre-release quality metrics and post-release quality are important directions of future research. Second, this study found that planning and tracking metrics for iteration were often used indicating a need to focus future research efforts on these areas. Third, use of metrics for motivating and enforcing process improvements can be an interesting future research topic. Fourth, the use of situative metrics in industrial agile context should be further investigated. Finally, the

dysfunctional use of metrics and the negative effects of metric use provide interesting future research topics.

To conclude, metrics can be used to improve aspects of agile software development but they can also have negative effects. One needs to have a holistic understanding of metrics in order to successfully apply them.

References

- D. J. Anderson. *Kanban*. Blue Hole Press, 2010.
- K. Beck and C. Andres. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.
- 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.
- B. Boehm and R. Turner. Using risk to balance agile and plan-driven methods. *Computer*, 36(6):57–66, 2003.
- C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346–7354, 2009.
- D. S. Cruzes and T. Dyba. Recommended steps for thematic synthesis in software engineering. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*, pages 275–284, 2011.
- T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9):833 – 859, 2008.
- N. E. Fenton and S. L. Pfleeger. *Software metrics: a rigorous and practical approach*. PWS Publishing Co., 1998.
- E.M. Goldratt. *The Haystack Syndrome: Sifting Information Out of the Data Ocean*. North River Press Publishing Corporation, 2006.
- R. B. Grady. *Practical software metrics for project management and process improvement*. Prentice-Hall, Inc., 1992.
- R. B. Grady. Successfully applying software metrics. *Computer*, 27(9):18–25, Sept 1994.

- 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.
- ISO/IEC. Iso/iec 25010 - systems and software engineering - systems and software quality requirements and evaluation (square) - system and software quality models. Technical report, 2010.
- C. Jones. *Applied software measurement: global analysis of productivity and quality*, volume 3. Mcgraw-hill New York, 2008.
- Staffs Keele. Guidelines for performing systematic literature reviews in software engineering. Technical report, EBSE Technical Report EBSE-2007-01, 2007.
- B. Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33:2004, 2004.
- B. Kitchenham. What’s up with software metrics? - a preliminary mapping study. *Journal of Systems and Software*, 83(1):37–51, January 2010.
- 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.
- J. R. Landis and G. G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174, 1977.
- 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.
- M. Poppendieck and T. Poppendieck. *Lean software development: An agile toolkit*. Addison-Wesley Professional, 2003.
- K. Pulford, A. Kuntzmann-Combelle, and S. Shirlaw. *A quantitative approach to software management: the AMI handbook*. Addison-Wesley Longman Publishing Co., Inc., 1995.
- S. Purao and V. Vaishnavi. Product metrics for object-oriented systems. *ACM Computing Surveys (CSUR)*, 35(2):191–221, 2003.
- K. Schwaber and M. Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002.

- K. Schwaber and J. Sutherland. The scrum guide. *Scrum. org*, July, 2013.
- G. A. F. Seber. *The Estimation of Animal Abundance and Related Parameters*. Blackburn Press, 2002.
- 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.
- J. P. Womack, D.T. Jones, and D. Roos. *The machine that changed the world: The story of lean production*. Simon and Schuster, 2007.
- 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. Medeiros 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, 2010a
- [S20] K. Petersen and C. Wohlin. Software process improvement through the lean measurement (spi-learn) method. *Journal of Systems and Software*, 83(7):1275–1287, 2010b
- [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 first search string was:

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"))

It found 512 hits 19 September 2013.

The second search string was:

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"))

It found 220 hits 7 November 2013.

The third search string was:

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 "ag-
ile modeling" OR scrumban OR kanban OR scrum OR "extreme program-
ming" OR xp) AND (measur* OR metric OR diagnosticOR monitor*)) AND
(LIMIT-TO(LANGUAGE, "English")) AND (LIMIT-TO(SUBJAREA, "BUSI"))
AND (EXCLUDE (SUBJAREA, "ENGI") OR EXCLUDE(SUBJAREA, "COMP"))

It found 42 hits 10 December 2013.

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 (2008).

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.1: Definitions of found metrics

Primary study	Metric	Definition
[S10]	# of defects	Issues found from quality assurance cycle including variance from expected behavior.
[S7]	# of defects found in system test	Number of defects found in system test.
[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 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 units required.
[S12]	Completed web pages	Completed web pages.
[S16]	Cost performance index	Not defined in primary study.
[S3]	Critical defects sent by customer	Not defined 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 implemented but some work is put into them.

[S20]	Inventory of requirements over time	Amount of requirements in specific work space 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 sub-process 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 burn-down	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 implementation 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.
