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

Eetu Kupiainen

What metrics are agile industrial teams using, why and how?

A systematic literature review

Master's Thesis
Espoo, Fix me June 18, 2011

DRAFT! — February 27, 2014 — DRAFT!

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: What metrics are agile industrial teams using, why and how? A systematic literature review			
Date:	Fix me June 18, 2011	**Pages:**	53
Major:	Software Engineering	**Code:**	T3003
Supervisor:	Prof. Casper Lassenius		
Instructors:	Juha Itkonen D.Sc. (Tech.) Mika Mäntylä D.Sc. (Tech.)		

Agile development methods are increasing in popularity, yet there are limited studies on the reasons and use of metrics in industrial agile development.

This paper presents results from a systematic literature review. Based on the study, metrics and their use is focused to the following areas: Iteration planning, Iteration tracking, Motivating and improving, Identifying process problems, Pre-release quality, Post-release quality and Changes in processes or tools. The findings are mapped against agile principles and it seems that the use of metrics supports the principles with some deviations.

Surprisingly, we find little evidence of the use of code metrics. Also, we note that there is a lot of evidence on the use of planning and tracking metrics. Finally, the use of metrics to motivate and enforce process improvements as well as applicable quality metrics can be interesting future research topics.

Keywords:	agile, metrics, measurement, systematic literature review
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Eetu Kupiainen		
Työn nimi: Mitä mittareita ketterät tiimit teollisuudessa käyttävät, miksi ja miten? Systemaattinen kirjallisuuskatsaus			
Päiväys:	Fix me 18. kesäkuuta 2011	**Sivumäärä:**	53
Pääaine:	Ohjelmistotuotanto ja - liiketoiminta	**Koodi:**	T3003
Valvoja:	Prof. Casper Lassenius		
Ohjaajat:	Tkt Juha Itkonen Tkt Mika Mäntylä		

Noh, mistäs teit dippas?

- Nooh, tutkin minkälaisia mittareita ketterät tiimit teollisuudessa käytti, miten ja miksi.

Okei, no mikäs on sen dipan tärkein anti?

- Noh, kategorisoin mittareiden käyttöä ja syitä, johon halukkaat voivat tarkemmin tutustua.

No miksi kukaan haluis noita ihmetellä?

- Mulla (timolla) on sellainen teoria, että on syitä, jotka aiheuttaa ohjelmistoprojektien epäonnistumisia. Mäppäsin löydettyjä mittareita em. syykategorioita vasten, joten voin sanoa että mitä mittareita kannattaisi käyttää jos haluaa ehkäistä tietyistä syistä johtuvia failureita.

- Lisäksi mä näytän miten suurempi osa on reaktiivisia mittareita.

- Lisäksi ideana oli, että pystyisin luomaan jonkinlaisen tärkeiden mittareiden luokittelun. . . Tämä on osoittautunut aika hankalaksi. Mulla on nyt yksittäisiä mittareita, joita on ylistetty - ja osaan myös sanottu miksi. Ne voin esittää. Lisäksi jos kvantitatiivisesti ajateltuna, näyttää siltä että effort estimaatit on tärkeitä agiilissa, koska niitä näkyi paljon. Toisinpäin ajateltuna, eli mitkä ei oo tärkeitä, niin vois sanoa et koodimittarit. Niitä ei ollut juuri yhtään.

Asiasanat:	ketterä, mittarit, mittaus, systemaattinen kirjallisuuskatsaus
Kieli:	Englanti

Acknowledgements

I want to thank my instructors Juha Itkonen and Mika Mäntylä for their invaluable feedback and tireless attitude towards my questions. Also, I want to thank Kim Dikert and Timo Lehtinen for relaxing and thoughtful chats.

Espoo, Fix me June 18, 2011

Eetu Kupiainen

Abbreviations and Acronyms

SLR Systematic literature review

WIP Work in progress

!Fixme **Poista koko abbreviations jos ei tuu montaa** Fixme!

Contents

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 that typically focuses on the effectiveness of a single metric. For example, Catal et al. review fault prediction metrics (Catal and Diri, 2009), 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 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, measurement research in the context of agile methods remains scarce.

The goal of this paper is to review the literature of actual use of software metrics in the context of agile software development. This study will lay 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 actions that the use of metrics can trigger.

This study covers the following research questions:

- RQ1: Why are metrics used?
- RQ2: What actions do the use of metrics trigger?
- RQ3: Which metrics are used?

- RQ4: What metrics are important?

1.1 Structure of the Thesis

This thesis is structured as follows. Chapter 3 describes how the SR was conducted. Chapter 4 reports the results from the study. Chapter 5 discusses about the findings and how they map to agile principles. Chapter 6 concludes the paper. !Fixme **Final structure** Fixme!

Chapter 2

Background

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 agile principles that give an idea what is agile development about. Popular agile development methods include Scrum (Schwaber and Beedle, 2002), Extreme Programming (Beck and Andres, 2004) and Kanban (Anderson, 2010).

2.2 Related work

Hartmann and Dymond (2006) don't provide any specific agile metrics but rather describes how agile metrics should be chosen and how they should be introduced to the organization. Also, they provide a set of heuristics for agile metrics.

!Fixme **tää jäi vähän laihaks** Fixme!

2.3 Systematic literature review

Systematic literature review is a research method originated from the field of medicine !Fixme **viite** Fixme!. The overarching idea is to aggregate and synthetisize existing knowledge regarding a research topic. This rigorous and audible evaluation method can facilitate theory development and point out gaps in research (Webster and Watson, 2002).

2.4 Causes for software project failures

2.5 Evidence based software engineering

2.6 Previous metric research

Chapter 3

Review method

Systematic review (SR) was chosen as 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

Kitchenham's guide for SRs (Kitchenham, 2004) was used as a basis for developing the review protocol. Additionally, a SR on agile development (Dybå and Dingsøyr, 2008) and a SR on SR (Kitchenham and Brereton, 2013) were used to further understand the challenges and opportunities of SRs. 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.

Table 3.1: Paper selection funnel

Stage	Amount of papers
Stage 1	774
Stage 2	163
Stage 3	29

Scopus database [1] was used to find the primary documents with automated search. Keywords include popular agile development methods and synonyms for the word metric. The search was improved incrementally in three phases because some key papers and XP conferences were not found initially. The search strings, hits and dates can be found from appendix A.

The selection of the primary documents was based on an 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 couldn't 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 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 work flow. 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 meetings. Also, selection of 7 papers was conducted by the second instructor with an almost perfect agreement, Kappa 1.0 (Landis and Koch, 1977).

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

3.3 Data extraction

Integrated coding was selected for data extraction strategy (Cruzes and Dyba, 2011). It provided focus to 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 measurement used?
- How is measurement used?
- Metric
- Importance related to 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 actions it causes or if it is described as important.
- Don't collect metrics that are only used for the comparison and selection of development methods.
- Don't collect metrics that are primarily used to compare teams.

Atlas.ti Visual QDA(Qualitative Data Analysis), version 7.1.x was used to collect and synthesize the qualitative data. Amount of found quotes per code can be seen in section 3.3.

To evaluate the repeatability of finding the same metrics, second instructor coded metrics from three papers. Capture-recapture method (Seber, 2002) was then used which showed that 90% of metrics were found.

A quality assessment form adopted from (Dybå and Dingsøyr, 2008) was used to evaluate the quality of each primary study. 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 the paper was for this study. The scale for the relevancy factor is:

Table 3.2: Amount of found quotes

Code	Amount of quotations
Why use this metric?	151
How is measurement used?	61
Metrics	108
Importance related to metric	45
Context	158

- 0 = doesn't contain any information regarding metrics and should be already 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 usage

3.4 Data synthesis

Data synthesis followed the steps recommended by Cruzes and Dyba (2011). Process started by going through all quotes within one code and giving each quote a more descriptive code describing the quote in high level. Then the descriptive codes were organized in groups based on their similarity. These groups were then given a high level code which are seen as categories in table 4.6.

Chapter 4

Results

This chapter presents the results from the systematic literature review. Table 4.1 shows the distribution of primary studies by publication channels. Table 4.3 lists the distribution of agile methods and table 4.4 lists the distribution of domains.

4.1 Overview of studies

4.1.1 Quality of studies

The perceived quality of the studies varied a lot (from 0 to 10). Even with many low quality studies they were included since they still provided valuable insight. For example in some cases experience reports can provide more valuable data than a high scoring research papers.

According to the assessment control group and reflexivity had lowest total scores while value for research, context and findings scored the highest.

4.2 Why were metrics used and how were they used?

Categories for the reasons and the use of measurements are listed in table 4.6. The following chapters will describe each category in more detail.

4.2.1 Iteration planning

Many metrics were used to support iteration planning. The metrics were used for task prioritization and scoping of the iteration.

Table 4.1: Publication distribution of primary studies

Publication channel	Type	#	%
Agile Conference	Conference	8	38
HICCS	Conference	3	14
ICSE	Workshop	2	10
XP Conference	Conference	2	10
Agile Development Conference	Conference	1	5
APSEC	Conference	1	5
ASWEC	Conference	1	5
Elektronika ir Elektrotechnika	Journal	1	5
Empirical Software Engineering	Journal	1	5
EUROMICRO	Conference	1	5
ICSE	Conference	1	5
ICSP	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

Table 4.2: Distribution of research methods

Research method	Amount
Multicase	2
Experience report	7
Singlecase	19
Survey	1

Table 4.3: Distribution of agile methods

Agile method	Amount
Scrum	15
XP	7
Lean	5
Other	5

Table 4.4: Distribution of domains

Domain	Amount
Telecom	10
Enterprise information system	7
Web application	4
Other	11

Many metrics were focused to help in the prioritization of the tasks for the next iteration [S8,S9,S11]. Prioritization of features was affected by a metric that measured the amount of revenue a customer is willing to pay for a feature [S11].

Effort estimation metrics were used to measure the size of the features [S6]. Furthermore, velocity metrics were used to calculate how many features is the team able to complete in an iteration [S23]. Knowing the teams' effective available hours was found useful when selecting tasks for an iteration [S3]. Velocity metrics were also used to improve the next iteration estimates [S16]. In one case, task's start and end date metric was used to point out interdependent tasks in the planning phase [S12].

4.2.2 Iteration tracking

Purpose of iteration tracking was to track how the tasks selected for the iteration were performed and that necessary modifications were done to the plan to complete the iteration according to schedule.

Metrics helped in monitoring, identifying problems, and predicting the end result by making it transparent to the stakeholders how the iteration is progressing. [S21,S27,S16,S5,S12,S7,S29,S30].

Progress metrics included number of completed web pages [S12], story completion percentage [S29] and velocity metrics [S5]. However, using velocity metrics had also negative effects such as cutting corners in implementing features to maintain velocity with the cost of quality [S6]. One qualitative progress metric was product demonstrations with customer [S30]. Measuring the completion of tasks enabled selecting incomplete tasks to the next iteration [S12].

When the metrics indicated, during an iteration, that all planned tasks could not be completed, the iteration was rescoped by cutting tasks [S16,S5,S17] or adding extra resources [S5,S17].

When there were problems that needed to be fixed, whether they were short or long term, the metrics helped in making decisions to fix them

Table 4.5: Quality assessment fo primary studies

Study	Rese arch	Aim	Con text	R.de sign	Samp ling	Ctrl. Grp	Data coll.	Data anal	Reflex iv- ity	Find ings	Value	Total	Rele vancy
[S1]	1	1	1	1	1	0	1	1	1	1	1	10	2
[S2]	0	0	0	0	1	0	0	0	0	0	1	2	2
[S3]	1	1	0	1	0	0	0	0	0	0	0	3	2
[S4]	0	0	0	0	1	0	0	0	0	0	1	2	3
[S5]	1	1	1	1	1	0	1	1	0	1	1	9	3
[S6]	0	0	1	0	1	0	0	0	0	0	1	3	2
[S7]	0	0	0	0	0	1	1	1	0	1	1	5	2
[S8]	0	0	0	0	0	1	0	0	0	1	1	3	3
[S9]	1	1	1	1	0	0	1	1	1	1	0	8	2
[S10]	0	0	1	0	1	1	0	0	0	1	1	5	2
[S11]	0	0	1	0	0	0	0	0	0	1	1	3	3
[S12]	0	0	1	0	0	0	0	0	0	0	0	1	3
[S13]	0	0	0	0	0	1	0	0	0	1	1	3	3
[S14]	0	0	0	0	0	0	0	0	0	0	0	0	2
[S15]	1	1	0	1	1	1	1	1	0	1	1	9	2
[S16]	1	0	1	0	1	0	0	0	0	0	0	3	2
[S17]	1	1	1	1	1	0	1	0	0	1	1	8	3
[S18]	1	1	1	1	1	0	1	1	0	1	1	9	3
[S19]	1	1	0	1	0	0	0	0	0	1	0	4	2
[S20]	1	1	1	1	1	0	1	1	1	1	1	10	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
[S31]	0	0	0	0	1	1	0	0	0	0	0	2	2
Total	16	15	20	15	19	7	14	13	7	21	24		

Table 4.6: Categories for measurement usage

Categories	Sources
Iteration planning	[S6,S23,S3,S8,S12] [S16,S9,S11]
Iteration tracking	[S21,S27,S16,S5,S12] [S4,S7,S6,S17,S29] [S30,S10,S25,S24,S23] [S8,S20]
Motivating and improving	[S29,S28,S13,S4,S3] [S23,S25,S27]
Identifying process problems	[S21,S29,S17,S13,S25] [S16,S19,S18,S31,S22]
Pre-release quality	[S14,S5,S29]
Post-release quality	[S19,S3,S7,S25]
Changes in processes or tools	[S13,S21,S18,S4,S25,S14,S26,S20]

[S25,S5,S21,S4]. It was possible to base decisions on data, not only use common sense and experience [S28]. Balance of work flow was mentioned as a reason for using metrics in multiple papers [S23,S19,S20,S8,S21,S5,S13]. Progress metrics were used to focus work on tasks that matter the most [S28], avoid partially done work [S24], avoid task switching [S24] and polishing of features [S28].Finally, open defects metric was used to delay a release [S10].

4.2.3 Motivating and improving

This section describes metrics that were used to motivate people and support team level improvement of working practices and performance.

Metrics were used to communicate different data about the project or product to the team members [S29,S28,S23,S25,S27]. Measurement data motivated teams to act and improve their performance [S28,S23,S3,S4,S13]. Some examples included fixing the build faster by visualizing build status [S13,S4], fixing bugs faster by showing amount of defects in monitors [S3] and increasing testing by measuring product size by automated tests that motivated team to write more tests [S28].

4.2.4 Identifying process problems

Metrics were often used to identify or avoid problems in processes and work flows. This chapter describes how metrics were used to spot problems.

There were multiple cases highlighting how metrics are used to identify or predict problems in order to solve or avoid them [S21,S29,S16,S19,S18,S31].

Sometimes there were work phases where no value was added, e.g., "waiting for finalization". This type of activity was called waste and was identified by using lead time. [S22]

Story implementation flow metric describes how efficiently a developer has been able to complete a story compared to the estimate. This metric helped to identify a problem with receiving customer requirement clarifications. [S13]

Creating awareness with defect trend indicator helped to take actions to avoid problems [S25]. One common solution to problems was to find the root cause [S13,S17].

4.2.5 Pre-release quality

Metrics in the pre-release quality category were used to prevent defects reaching customers and to understand what was the current quality of the product.

Integration fails was a problem to avoid with static code check metrics [S14]. Moreover, metrics were used to make sure that the product is sufficiently tested before the next step in the release path [S14,S5]. Additionally, making sure that the product is ready for further development was mentioned [S7].

Some metrics forced writing tests before the actual code [S29]. Technical debt was measured with a technical debt board that was used to facilitate discussion on technical debt issues, see fig. 4.1 [S4].

4.2.6 Post-release quality

Metrics in post-release quality deal with evaluating the quality of the product after it has been released.

Customer satisfaction, customer responsiveness, and quality indicators were seen as attributes of post-release quality. Some metrics included customer input to determine post-release quality [S19,S7,S3] while other metrics used pre-release data as predictors of post-release quality [S25,S19,S7]. Customer related metrics included, e.g., defects sent by customers [S3], change requests from customers [S19] and customer's willingness to recommend product to other potential customers [S7]. Quality prediction metrics included defect counts [S19], maintenance effort [S25] and deferred defect counts [S7].

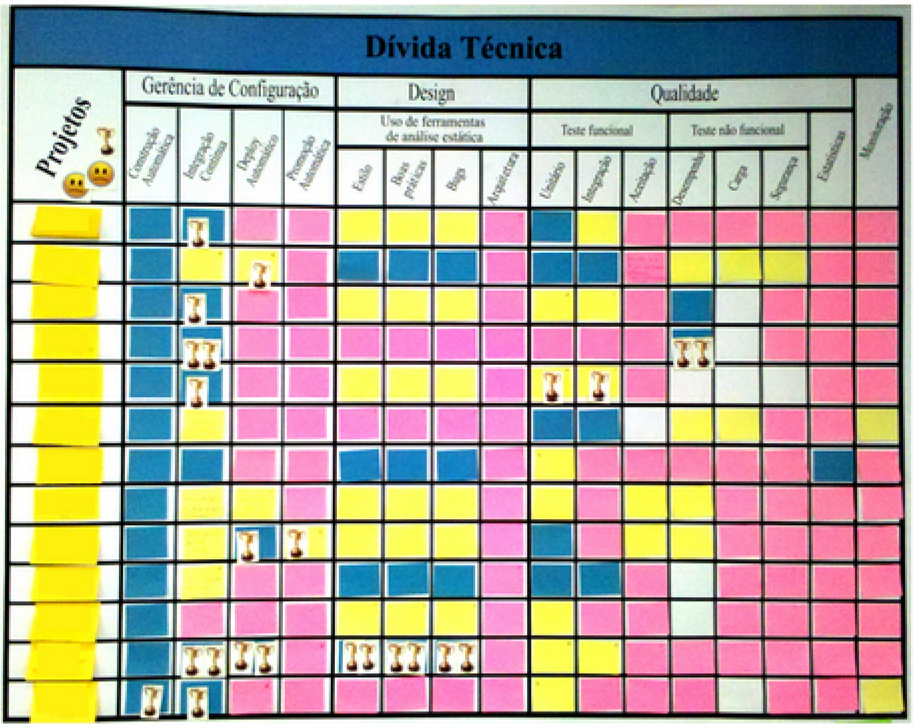

Figure 4.1: Technical debt.

4.2.7 Changes in processes or tools

This chapter describes the reported changes that applying metrics had for processes and tools. The changes include changes in measurement practices, development policies, and the whole development process.

The successful usage of sprint readiness metric and story flow metric changed company policy to have target values for both metrics as well as monthly reporting of both metrics by all projects [S13].

At Ericsson by monitoring the flow of requirements metric they decided to change their implementation flow from push to pull to help them deliver in a more continuous manner. Also, based on the metric they added an intermediate release version to have release quality earlier in the development cycle. [S20]

Changes to requirements management were also made based on lead time in other case at Ericsson. Analysing lead time contributed to delaying technical design after purchase order was received, providing customer a rough estimate quickly and merging the step to create solution proposal and technical design. [S18]

Problem with broken build, and the long times to fix the build, led to measurements that monitor and visualize the state of the build and the time it takes to fix it [S4,S13,S14].

Also, additional code style rules were added to code check-in and build tools so that builds would fail more often and defects would get caught before release [S13,S14].

Similarly, testing approaches were changed based on flow metrics. Using lead time led to that integration testing could be started parallel to system testing [S18]. Also, throughput of a test process showed insufficient capability to handle the incoming features, which led to changing the test approach [S26].

4.3 Why were metrics used?

4.3.1 Identify problems

Cumulative number of work items over time metric was used to identify bottlenecks in the development process. Similarly, lead time was used to identify waste of waiting [S22].

Also, value stream maps (VSM) were used to spot waste in the development process. [S18]

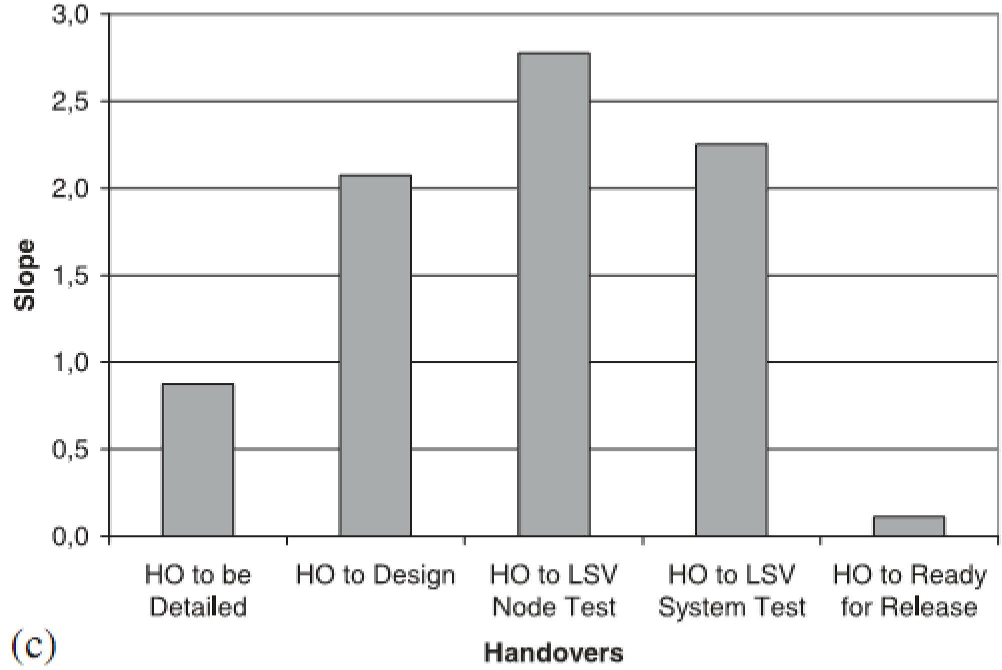

Figure 4.2: Handovers [21]

Monitoring Work-in-progress (WIP) was used to spot blocked work items and also the development phase where the blockage occurred. [S2]

Defect trend indicator was used to provide the project manager an ISO/IEC 15939:2007 compatible indicator for problems with the defect backlog. Basically, the indicator showed if the defect backlog will increase, stay the same or decrease in the coming week. The project manager could then use the info to take necessary actions to avoid possible problems. [S25]

Monitoring schedule and costs with a dashboard allowed to spot for improvement opportunities at OCLC. [S31]

Cost types, rate of requirements over phases and variance in handovers were for process improvement by spotting bottlenecks and uneven requirement flows. [S21]

Number of work items per phase and leadtime was used to spot instabilities in the process. If a measured value was outside the control limits it was defined an instability. This enabled process improvement. [S20]

Schedule performance index and cost performance index were used to monitor for deviances in the project's progress and providing early sign if something goes wrong. [S16]

Measuring story flow percentage allowed identification of waste related to context shifts. [S13]

Developers at Avaya had issues with the 80/20 rule, where the last 20% of iteration takes the longest. With the metrics that their tool T3 provides (e.g Story percent complete) they were able to see the early symptoms of various problems that can cause delays, and react early. [S29]

4.3.2 Planning

Marking tasks done and undone made it possible to take undone tasks for the next iteration. Developers would mark expected end date for task so the next one can plan their own work as effectively as possible thus reducing idle time. Leaders use estimates to check the if the planned scope would be possible to complete during the next iteration. [S12]

Effort estimates were used to prioritize the features for next release and used as basis for resourcing. [S9]

Velocity was used to improve effort estimates for next planning session, and that way it was easier to understand how large can the scope be for the next iteration. [S16]

At WMS, they used pseudo-velocity and average velocity to plan their releases. [S23]

At Verisign Managed Security Services, they used many attributes to prioritize their backlog and revenue per customer was described in more detail. [S11]

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]

Stories and their effort estimates were used as the fundamental units of development for the iteration. [S6]

Story estimates were used to understand the iteration where different features will be completed. [S29]

Lead time was used to understand if all planned corrections can be completed before release date. [S24]

Teams use effort estimates to prioritize activities based on relative value and relative effort. [S8]

At Timberline, they broke down requirements into smaller pieces that were estimated in effort to understand what skills are needed to complete the work. Also, they used Kano analysis as a 'voice of customer' so that prioritization decision could be based on facts instead of political power. [S17]

4.3.3 Progress tracking

4.3.3.1 Balance workflow

Inventory of requirements over time was used to identify large handovers of requirements that would cause overload situations. The aim was to have steady flow of requirements. [S20]

Variance in handovers was used to guarantee that requirements would flow evenly. [S21]

Operations was overloaded so they decided to start evaluating incoming work with Ops story points to level the workload. [S8]

People should be respected by having balanced workload to avoid overload situations. Measuring number of requirements per phase would help noticing peaks. [S22]

Timberline tried to pace work according to customer demand. However, too much work was pushed to development, which caused many problems. They started using common tempo time to make sure there would be balance of workflow. [S17]

Mamdas was measuring 'pulse' metric, which measured how often code is committed to main trunk. The point was to avoid people from committing only at the end of the iteration and instead integrate early and often. [S5]

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]

4.3.3.2 Project progress

Completed web pages was used as a measure of progress. [S12]

Number of written and passed unit tests was used as a measure of progress in terms of completed work. [S5]

Breaking down tasks to 'kits' between 2 to 5 days enabled progress monitoring. [S17]

Set of metrics(burndown, 'pulse', number of written and passed unit tests) was developed to manage risks and provide timely progress monitoring. [S27]

Story percent complete metric is used to give assessment of progress. [S29]

Release burndown shows project trends and can be used to predict completion date. Release burndown also reflects addition or removal of stories. [S16]

Cost types, rate of requirements over phases and variance in handovers were used to provide overview of progress. [S21]

4.3.3.3 Visibility

Metrics(burndown, 'pulse', number of written and passed unit tests) were used to replace individual perception with facts. [S27]

Defect trend indicator was used to monitor defect backlog and spread the information to project members. [S25]

Cycle time metrics were used to let the team track their own performance. [S23]

Story percent complete metric were generated automatically when tests were run and thus kept everyone on the same page and eliminated schedule surprises. The metric results were required to be reported periodically as well. [S29]

Technical debt board was used to make technical debt issues visible and easier to manage. [S4]

Release burndown made the correlation clear between work remaining and team's progress in reducing it. [S16]

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]

4.3.3.4 Report to upper management

Metrics(burndown, check-ins per day, number of written and passed unit tests) were used to communicate progress to upper management [S5] and ensure good progress to external observers and ensure that key risks were under control [S28].

4.3.3.5 Iteration goal assuring

There was a need for simple indicator that would quickly tell if project is under control. Common compo time was used to understand if project was in target for delivery. [S17]

Monitoring WIP was used predict lead time which in turn will predict project schedule. [S2]

Sprint burndown was used to tell the team if they were on track regarding the sprint commitments. [S7]

Burndown was used to see if the team could meet their goals, and if not what could be done. [S5]

Story flow percentage was used so that developer could finish a story in a steady flow. [S13]

Burndown was used to mitigate the risk where developers spend too much time perfecting features over finishing all tasks of the iteration. [S28]

4.3.3.6 Misc

A team at NASA Ames Reserch center didn't want to spend resources on estimating features and instead focused on designing and developing their software solution. Every six weeks they demonstrated their progress to customer with working code. [S30]

Component level burndown was used to decide on resource mobility.[S5]

Monitoring and managing risks was the reason for using metrics (burndown, check-ins per day and number of written and passed unit tests) [S5,S27,S28].

4.3.4 Improve quality, understand level of quality

Faults per iteration were used to measure the product's quality. Using number of written and passed unit tests dealt with decreasing the risk that the product would be un-thoroughly tested. Similarly, number of written and passed unit tests was used to make sure regression tests are ran and passed every iteration. [S5]

At T-Systems International, they used a set of metrics(broken build, number of unit tests, test coverage, test growth ratio, violations of static code analysis) to improve project's internal software quality. [S14]

Test coverage was used to evaluate how well was the code tested. However, in Brown-field(legacy) projects it's better to measure test-growth-ratio since there might be not be that many tests in the existing code base. [S14]

Broken build was measured to prevent defects reaching production environment. [S14]

Violations of static code analysis was used to prevent the existence of critical violations. [S14]

Technical debt board was used to reduce technical debt. [S4]

Build status was used to motivate people to fix the build as fast as possible. [S4]

Fix time of failed build was used to motivate people to fix it faster. [S13]

Work in progress was measured so it could be minimized. Otherwise there would be problems e.g when there would be lot of untested code. [S17]

Number of change requests from customer was used as indicator customer satisfaction. [S19]

Maintenance effort was used as an indicator of overall quality of the release product. [S19]

Governance mechanisms, which included a set of metrics (burndown, check-ins per day and number of written and passed unit tests), were used to increase product quality. [S28]

Check-ins per day was used to prevent people from integrating only late in the integration and instead encourage constant integration throughout the iteration. [S28]

Critical defects sent by customers were tracked and fixed to prevent losing customers. [S3]

Predicted number of defects was used to plan the removal of defects. If the removal of defects would not be well planned it could cause delays for the release and thus increase costs for the project. [S25]

Net Promoter Score was measured because it was thought to be a pure measure of success, since it's measured from the customers. [S7]

Defects found in system test was used to measure the quality of software delivered to system test process. [S7]

Defects deferred was used to predict the quality customers would experience. [S7]

Number of maintenance requests was used as an indicator of built in quality. [S22]

Story percent complete metric supports test driven development. [S29]

4.4 What kind of effects did the use of metrics have?

4.4.1 Create improvement ideas

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

When a waste of extra process(requirement would wait for long time before full specification) was identified in requirement specification with the help of lead time, processing time and queue time metrics, a solution idea was created where a quick high level proposal would be sent to customer without the need for in-depth specification. Customer could then use the high level proposal to evaluate if they want to pursue that requirement further. Similarly, two requirement specification phases could be combined when another waste of extra process was identified in requirements specification phase. Additionally, lead time could be decrease by increasing the collaboration with the market unit and the development unit. Similarly, there was a waste of waiting in design which could be improved by starting real work only when the purchase order is received, not when requests are received. [S18]

Same metrics as above were used to identify waste of waiting in testing phases. The improvement suggestion was to provide earlier beta version and making testing phases parallel. Many of the improvement ideas came from meetings where the value stream maps (VSM) were used as a base for discussion. [S18]

Cost types, rate of requirements over phases and variance in handovers were used to identify bottlenecks at Ericsson. They noticed that focusing on deadlines caused a lot of requirements to be transferred to system test phase close to the deadline. The improvement suggestion was to focus more on continuous delivery instead of focusing on market driven deadlines. Furthermore, Kanban was suggested as a development method to accomplish the continuous delivery capabilities. [S21]

Throughput and queue time metrics were used to identify bottleneck in network integration test phase which lead to using other testing practices in future projects. [S26]

Rate of requirements over time was used to identify problems in the development process. One improvement suggestion was to change from push to pull-approach so that team can adjust the workload to enable 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. [S20]

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. This led to other departments trying to decrease their products' Ops story points to enable faster releases. The reduction in story points was possible by creating hot deployment strategies and providing better documentation. [S8]

Mamdas, an Israel Air force IT department, were using burndown to follow their progress. However, when they noticed that work remaining wasn't 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. [28]

When story implementation flow metric showed a drop and project managers complained about clarifications about features from customer were late, a root cause analysis meeting was held. 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%, and both of the metrics need to be reported montly. [S13]

Root cause analysis was also conducted to decrease the amount of bounce backs. [S17]

A team noticed their velocity estimations were inaccurate which led to dividing work items into smaller pieces to improve the accuracy of the estimates. [P10]

The reasons for the values of metrics(burndown, check-ins per day, number of written and passed unit tests) were discussed in iteration summary meeting because it can be hard to analyze metrics without understanding the context [S27]. Similarly, 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, if the values of number of work items were outside the control limits one could discuss with the developers about the workload. [S22]

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]

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

4.4.2 Planning actions

Product owners use lead time to schedule high priority features and plan demo dates with customers. Velocity / 2 metric is used as scoping tool for a release. The team has enough work not to sit idle, but still flexible enough to allow high priority bug fixes to be fixed. [S23]

Velocity was used to define minimum delivery level for the iteration where 'must have' requirements are assigned, and a stretch goal where lower priority requirements are assigned. [S2]

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

Effort estimates are used to scope the iteration and if there are tasks that cannot be completed before release date then they are excluded from the backlog. [S12]

4.4.3 Reactive actions

When component level burndown was used to notice that a component was behind schedule, resources were added and scope was reduced for the release. [S5]

If team effectiveness is not high enough to complete tasks, resources from other teams can be used. Other actions that were suggested were reduction of tasks and working overtime. [S3]

Release burndown showed that work remaining was not decreasing fast enough so the scope of the release was decreased. [S16]

Monitoring cycle times revealed high time consumption on manual testing. The cause was an unmotivated person who was moved to writing automated test scripts which he liked more. [S17]

If common compo time would show too much work, then tasks would be cut or more resources would be added. Similarly, if work balance is uneven, some roles have too much work while some don't have enough, people were trained in multiple roles, e.g customer support did testing and documentation engineers were taught how to input their material into the system. [S17]

Number of defects were used to delay a release when too many defects were noticed in a QA cycle. [S10]

Expected date of completion was used so that other team members could plan their own work. [S12]

Quality manager interpreted 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, when the issue was discussed in a reflection meeting 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 do them on their own as well, and later a dedicated tester was assigned to write the tests. [S28]

4.4.4 Motivate people

Number of defects was shown in monitors in hallways and that motivated developers to fix defects. [S3]

Total reported defects and test failure and success rate was also show throughout the organization which motivated people to avoid problems and also fix problems fast. [S3]

Build status was visible in minutes after commits, which helped to create a culture to react with high priority to broken builds. This helped to keep the main branch to be closer to deployable state at all times. [S4]

After meetings where technical debt categories were analyzed, team members agreed which categories they would focus in decreasing until the next meeting. Also, team members sought help from the architecture team for reducing technical debt, e.g by implementing automatic deployment or improving source code unit testability. [S4]

Measuring the number of written and passed unit tests changed teams behaviour to write more tests. [S5]

Fix time of failed build metric improved people's mindset to react to problems immediately. The metric was declared mandatory for all projects. Also, the reasons for long fix times are investigated. [S13]

Violations of static code analysis cause developers to immediately fix the issue because the violations can cause a broken build status. Additionally, developers get faster feedback on their work. Furthermore, developers have more confidence in performing major refactorings with the safety net the violations of static code analysis metric provides. [S14]

Metrics were also used to prevent harmful behaviour such as cherry picking features that are most interesting to the team. Measuring work in progress (WIP) and setting WIP limits prevented cherry picking by enforcing only two features at a time and thus preventing them from working on lower priority but more interesting features. [S17]

Defect trend indicator created crisis awareness and motivated the developers to take actions in hopes to avoid possible problems. [S25]

However, using velocity metric had also negative effects such as cutting corners in implementing features to maintain velocity with the cost of quality [S6].

4.5 Metrics & categorization

Metrics are listed by primary study in appendix, see appendix D.

Metrics could help in decreasing software project failures due to prioritization and resource & schedule issues in iteration tracking. Metrics here are all reactive.

Iteration tracking metrics could help in monitoring issues that lead to project failures. Metrics are mostly reactive.

Metrics that are used to motivate and improve people can be used to solve issues related to values & responsibilities and company policies that lead to project failures. Metrics are both reactive and proactive.

Metrics that are used to point identify problems can be used to prevent Method causes for failures. Metrics are almost all proactive.

Metrics that are used to improve or understand pre-release quality can be used to prevent failures that are caused by value & responsibility, task output and existing product related issues. Metrics are mostly reactive.

Metrics that are used for post-release quality can be used to prevent failures that are caused by customers' and users' opinions. Metrics are mostly proactive.

In general, used metrics were more reactive than proactive.

Table 4.7: Metrics categorized by causes for failures

Causes for failures	**My categories**							
	Iteration planning	Iteration tracking	Motivating and improving	Identifying process problems	Pre-release quality	Post-release quality	Cannot Say My Category	Total
People								
Instructions & Experience	1	0	0	1	0	0	0	2
Values & responsibilities	0	0	2	2	3	0	0	7
Cooperation	1	0	0	0	0	0	0	1
Company Policies	0	0	3	0	0	0	0	3
Tasks								
Task Output	1	0	0	2	4	0	0	7
Task Difficulty	0	0	0	1	0	0	0	1
Task Priority	5	1	0	0	0	0	0	6
Methods								
Work Practices	0	0	0	7	2	0	0	9
Process	0	1	1	8	0	0	0	10
Monitoring	0	10	0	1	4	0	0	15
Environment								
Existing Product	0	0	0	0	6	0	0	6
Resources & Schedules	11	3	0	0	0	0	0	14
Tools	0	0	0	0	0	0	0	0
Customers & Users	0	1	0	0	0	7	0	8
Cannot Say Cause Category	0	2	0	0	0	0	15	17
Total	19	18	6	22	19	7	15	106

In general, it seems that metrics are used the most to prevent project failures that are caused by Methods, then by Environment and not so much about People or Tasks.

4.6 Important metrics in terms of statements

This section describes metrics that 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. Also, a minimum 60% value for flow was identified. Similarly, velocity for elaborating features should be as fast as velocity of implementing features. Also, they said using both metrics *"drive behaviors to let teams go twice as fast as they could before"*. [S13]

Net Promoter Score was said to be *"one of the purest measures of success"* [S7].

According to a survey, 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. [S1]

Story percent complete metric was considered valuable since it embraces test driven development - no progress is made before test is written. Also, percent complete metric is considered more accurate than previously used metric. Moreover, it gives normalized measure of progress compared to developer comments about progress. Additionally, story percent complete metric leverages existing unit testing framework and thus requires only minimal overhead to track progress. Team members seemed to be extremely happy about using the metric. [S29]

Pseudo-velocity was considered essential for release planning [S23].

In an agile survey [S1], project success had significant positive relationship with the following metrics: team velocity, business value delivered, running testing features, defect count after testing and number of test cases. However, there were no detailed descriptions of these metrics.

Effort estimates were consider important in release planning especially in terms of prioritization [S9].

Burndown was valuable in meeting sprint commitments [S7]. Similary, managers said burndown was important in making decisions and managing multiple teams [S5]. However, developers didn't consider burndown important [S5].

Top performing teams at Adobe estimated backlog items with relative effort estimates [S7].

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

At another case at Ericsson Value Stream Maps (VSM) were used to visualize problem areas and possible improvements. Practitioners valued how the maps were easy to understand. Metrics that were used to build VSM were lead time, processing time and queue time. [S18]

Defects deferred 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. Key attributes of the metrics were sufficient accuracy and ease of use. [S26]

Technical debt board that visualized the status of technical debt in categories was considered important because it gave a high level understanding of the problems and it was then used to plan actions to remove technical debt. It was proven to be useful in their context. [S4]

The following metrics were consider very useful in agile context: number of unit tests, test coverage, test-growth ratio and broken builds. The benefit for the number of unit tests is not well described except that it provides *"first insights"*. Test coverage provides info on how well the code is tested. Test-growth ratio is useful in projects where old codebase is used as basis for new features. Fixing broken builds prevents defects reaching customers. [S14]

4.6.1 Important metrics based on amount of evidence

Effort estimation metrics were used in many cases, so it could be perhaps said they are important since they provide scoping and prioritization.

Chapter 5

Discussion

5.1 Implications for practice

To provide implications to practice the findings are mapped to the principles of agile software development Beck et al. (2007) categorized by Patel & al. Patel et al. (2006). For each paragraph the naming by Patel et al. is used and references to the agile practices is provided by numbers.

Communication and Collaboration (principles 4 and 6) was reflected in metrics that motivated a team to act and improve, see section 4.2.3. Also, progress metrics were used to communicate the status of the project to the stakeholders, see section 4.2.2.

Team involvement (5,8) was reflected in metrics that motivated team to act and improve, see section 4.2.3. Also, to promote sustainable development metrics were targeted to balance the flow of work, see section 4.2.2.

Reflection (12) was visible in metrics that were used to identify problems and to change processes, see section 4.2.4 and section 4.2.7.

Frequent delivery of working software (1,3,7) was directly identified in one paper, where the team measured progress by demonstrating the product to the customer [S30]. Additionally, there were cases where e.g. completed webpages [S12] were the primary progress measure. Also, many metrics focused on progress tracking and timely completion of the iteration, see section 4.2.2. However, some other measures from section 4.2.2 show that instead of working code agile teams followed completed tasks and velocity metrics.

An integral part of the concept of working software is measuring post-release quality, see section 4.2.6. 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 each iteration, see section 4.2.1. Additionally, different metrics helped keeping the internal quality of the product high throughout the development which then provided safe development of modifications from new ideas, see section 4.2.5.

Design (9,10,11) was seen in focus to measuring technical debt and using metrics to enforce writing tests before actual code, see section 4.2.5. Additionally, the status of build was continuously monitored, see section 4.2.7. However, the use of velocity metric had a negative effect on technical quality, see section 4.2.2. Many metrics focused on making sure that the right features were selected for implementation, see section 4.2.1, thus avoiding unnecessary work.

There were also metrics, or their usage, which were not agile in nature. E.g., maintaining velocity by cutting corners in quality instead of dropping features from that iteration [S6]. Also, adding people to project to reach a certain date [S5, S17] doesn't seem that agile compared to removing tasks. Adding people can have a negative impact to progress, considering the lack of knowledge and training time required. Moreover, the use of dates to plan interdependent tasks is not agile in nature [S12]. Instead, interdependencies should be visible in choosing the tasks to appropriate iterations. Also, 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.

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.

5.1.1 Comparison to prior studies

Only few papers have broadly studied the reasons for software metrics use in the context of agile software development. Hartmann and Dymond (2006) also highlight process improvement as one of the reasons for measurement in their agile metrics paper. Also, they emphasize that creation of value should be the primary measure of progress - which was also seen in this study.

Korhonen (2009) found in her study that traditional defect metrics could be reused in agile context - if modified. Defect metrics were also used in many of the primary studies.

Kitchenham's mapping study (Kitchenham, 2010) identified several code metrics in academic literature. However, in this study almost no evidence of code metric use in the industrial context was found. Maybe it is time to re-evaluate the need for code metrics research if industry doesn't seem to use them.

5.2 Limitations

The large shares of specific application domains in the primary documents is a threat to external validity. Seven out of 29 studies were from 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 selection 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.

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, refine the method and tools.

Some data from low scoring papers, e.g [S3], are not justified very detailed which could cause incorrect interpretations.

Chapter 6

Conclusions

This study presents the results from a systematic literature review from 29 !FIXME **numero** FIXME!primary studies. According to the researcher's knowledge there is no previous systematic reviews of measurement use in the context of industrial agile software development. This study classifies and describes the main measurement types and areas that are reported in empirical studies. This study provides descriptions of how and why metrics are used to support agile software development.!FIXME **lisää important metrics ja metrics categories** FIXME! This study also analyzed how the presented metrics support the twelve principles of Agile Manifesto (Beck et al., 2007).

The results indicate that the reasons and use of metrics is focused on the following areas: Iteration planning, Iteration tracking, Motivating and improving, Identifying process problems, Pre-release quality, Post-release quality and Changes in processes or tools.

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

Finally, this study identified few propositions for future research on measuring in agile software development. First, in the academia lot of emphasis has been given to code metrics yet this study found little evidence of their use. Second, the applicable quality metrics for agile development and the relationship of pre-release quality metrics and post-release quality are important directions of future research. Third, this study found that planning and tracking metrics for iteration were often used indicating a need to focus future research efforts on these areas. Fourth, use of metrics for motivating and enforcing process improvements can be an interesting future research topic.

!FIXME **lisää important metrics ja metric categories** FIXME!

References

D.J. Anderson. *Kanban*. Blue Hole Press, 2010.

Kent Beck and Cynthia Andres. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2007.

Cagatay Catal and Banu Diri. A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346–7354, 2009.

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

Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50 (910):833 – 859, 2008.

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.

Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33:2004, 2004.

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

Barbara Kitchenham and Pearl Brereton. A systematic review of systematic review process research in software engineering. *Information and Software Technology*, 55(12):2049–2075, 2013.

Kirsi Korhonen. Migrating defect management from waterfall to agile software development in a large-scale multi-site organization: A case study. In Pekka Abrahamsson, Michele Marchesi, and Frank Maurer, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 31 of *Lecture Notes in Business Information Processing*, pages 73–82. Springer Berlin Heidelberg, 2009. URL `http://dx.doi.org/10.1007/978-3-642-01853-4_10`.

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.

Sandeep Purao and Vijay Vaishnavi. Product metrics for object-oriented systems. *ACM Computing Surveys (CSUR)*, 35(2):191–221, 2003.

Ken Schwaber and Mike Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002.

G.A.F. Seber. *The Estimation of Animal Abundance and Related Parameters*. Blackburn Press, 2002. URL `http://books.google.fi/books?id=bnGaPQAACAAJ`.

Jane Webster and Richard T Watson. Analyzing the past to prepare for the future: Writing a literature review. *Management Information Systems Quarterly*, 26(2):3, 2002.

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 cmmiÂ® process improvement at microsoft corporation. In *Proceedings - AGILE Confernce 2005*, volume 2005, pages 193–201, Denver, CO, 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] Paulo Sérgio Medeiros dos Santos, Amanda Varella, Cristine Ribeiro Dantas, and Daniel Beltr ao 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] VS. 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] PMahnic and N Zabkar. Measuring progress of scrum-based software projects. *Electronics and Electrical Engineering*, 18(8):73–76, 2012

[S17] S. 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. ISSN 17466474

[S18] K. 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. ISSN 13823256

[S20] K. Petersen and C. Wohlin. Software process improvement through the lean measurement (spi-leam) method. *Journal of Systems and Software*, 83(7):1275–1287, 2010b. ISSN 01641212. cited By (since 1996)6

[S21] K. Petersen and C. Wohlin. Measuring the flow in lean software development. *Software - Practice and Experience*, 41(9):975–996, 2011. ISSN 00380644

[S22] R. 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. ISSN 18651348

[S23] M. 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] Miroslaw Staron, Wilhelm Meding, and Bo 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. ISSN 03029743

[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

[S31] D. Tudor and G.A. Walter. Using an agile approach in a large, traditional organization. In *Proceedings - AGILE Conference, 2006*, volume 2006, pages 367–373, Minneapolis, MN, 2006

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 diagnosticOR 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 "agile modeling" OR scrumban OR kanban OR scrum OR "extreme programming" 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 don't contain empirical data from industry cases.
- Papers that are not in English.
- Papers that don't 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 don't 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 measurements are only used for the research. For example author measures which agile practices correlate with success.

Appendix C

Quality assesment questions

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

Metric distribution by primary studies

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 in categories, build status, technical debt in effort
[S5]	Burndown, check-ins per day, number of automated test steps, faults per iteration
[S6]	Velocity, story estimates
[S7]	Burndown, story points, # of open defects, defects found in system test, defects deferred, net promoter score
[S8]	Story points, task effort, velocity, operation's velocity
[S9]	Effort estimate, actual effort
[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]	Broken build, 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, number of work items per phase
[S21]	Rate of requirements per phase, variance in handovers, requirement's cost types
[S22]	# of requirements per phase, lead time
[S23]	Work in progress, average velocity, cycle time, pseudo velocity
[S24]	Lead time, work in progress, # of days in maintenance, # of days to overdue, reported hours on CSR
[S25]	Number of features developed in release, queue, throughput
[S26]	Defect trend indicator, # of defects in backlog, predicted # of defects
[S27]	Burndown, check-ins per day, number of automated test steps
[S28]	Burndown, # of new defects, number of written and passed tests, task estimated vs actual time, time reported for overhead activities, check-ins per day
[S29]	Story estimate, story complete percentage
[S30]	Progress as working code
[S31]	Costs, schedule