EXAMENSARBETE

The Right Way to do IT

Waste Identification and Reduction at Software Providers

- a Case Study at Infor M3

Adam Johansson
Jonathan Ryen

Civilingenjör, Industriell ekonomi
2017

Luleå tekniska universitet
Institutionen för ekonomi, teknik och samhälle

LULEÅ TEKNISKA UNIVERSITET

The Right Way to do IT

Waste Identification and Reduction at Software Providers - a Case Study at Infor M3

Författare:

Adam Johansson

Jonathan Ryen

Handledare:

Henrik Johansson, Infor Sweden AB

Erik Lovén, Luleå tekniska universitet

Examensarbete för Civilingenjör i Industriell Ekonomi utfört inom ämnesområdet kvalitetsteknik vid Luleå tekniska universitet och Infor Sweden AB

Luleå – 2017-06-27

Acknowledgments

We would like to acknowledge our supervisor at Infor, Henrik Johansson, for being an extraordinary mentor during this thesis. Henrik has sacrificed a lot of his precious time, only to ensure that this thesis would be completed in a satisfactory way for both us as authors and Infor as a case company.

Moreover, we would like to thank Erik Lovén as the academic supervisor during the study for being accessible and willing to answer our questions during the thesis.

Luleå, June 2017

Jonathan Ryen Adam Johansson

Abstract

When delivering Software as a Service (SaaS), increased challenges regarding responsiveness and flexibility are introduced to software providers. In order to address these challenges and attain customer satisfaction, the Lean Software Development principle *Eliminate waste* can be applied.

One organization that has recognized the potential of applying the *Eliminate waste* principle when developing software is the Integration & BI unit of the Enterprise Resource Planning system provider Infor M3. However, the implementation of the *Eliminate waste* principle at the Integration & BI unit's five teams is still in an early stage. Consequently, the intended purpose of the thesis was to identify waste and suggest approaches to reduce waste at the case organization, Infor M3 Integration & BI.

In order to collect the in-depth knowledge required, the thesis utilized a qualitative case study methodology, whereby a literature review, interviews and observations were conducted. The literature review created a foundation of knowledge regarding waste in software development, that subsequent culminated as a basis for the analysis and recommendations. It could be concluded that the subject of waste identification and reduction in software development is in an early stage, largely driven by practitioners, with few verifying studies that support the subject's applicability. However, by utilizing a waste categorization model various wastes could be identified at all of Integration & BI's teams during the interviews, whereupon *Partially done work*, *Delays*, *Task switching* and *Relearning* was considered as the most prominent wastes. Moreover, it could be established that one team had developed successful approaches that eliminates much of the team's waste whilst the other teams' approaches were generally deficient.

In order to more successfully reduce waste, the Integration & BI unit is suggested to create awareness of the concept of waste within the unit. The teams need a common definition and an increased understanding of waste in order to reach this awareness. Additionally, the unit is suggested to use more comprehensive indicators, like cumulative flow diagram, in order to facilitate identification and root-cause analysis of waste. Lastly, the unit is recommended to reduce waste by continuous improvements with activities structured as a PDSA-cycle.

Sammanfattning

Programvaruföretag som levererar Software as a Service (SaaS) står inför ökade utmaningar med avseende på receptivitet och flexibilitet. För att bemöta dessa utmaningar och uppnå hög kundnöjdhet kan Lean Software Development principen *Eliminera slöseri* appliceras.

En organisation som har insett potentialen av att tillämpa principen *Eliminera slöseri* vid utveckling av programvara är enheten Integration & BI hos affärssystemleverantören Infor M3. Implementeringen av principen vid enhetens fem utvecklingslag är emellertid fortfarande i ett tidigt skede. Avsikten med examensarbetet var således att identifiera slöseri och föreslå tillvägagångssätt för att minska slöseri hos fallorganisationen, Infor M3 Integration & BI.

För att samla in en djup förståelse om ämnet och fallföretaget utnyttjade examensarbetet en kvalitativ fallstudiemetodik, där en litteraturstudie, intervjuer och observationer genomfördes. Litteraturstudien skapade kunskap angående slöseri inom mjukvaruutveckling, vilken senare skapade en grund för både analys och rekommendationer. Slutsatsen att ämnet rörande identifiering och minimering av slöseri inom mjukvaruutveckling är i ett tidigt skede kunde göras i och med att ämnet i stor utsträckning drivs av yrkesutövare och inte av akademin. Dessutom finns det få vetenskapliga studier som verifierar ämnets applicerbarhet. Genom att använda en modell för att kategorisera olika slöseri inom mjukvaruutveckling kunde dock olika typer av slöseri identifieras hos samtliga av Integration & BIs utvecklingslag, varpå *delvist gjort arbete, förseningar*, *uppgiftsbyte* och *återinlärning* betraktades som de mest framträdande. Dessutom kunde det fastställas att ett av utvecklingslagen etablerat framgångsrika tillvägagångssätt för att eliminera slöseri undertiden de andra utvecklingslagens metoder generellt var bristfälliga.

För att mer framgångsrikt minska slöseri, föreslås enheten Integration & BI att använda en gemensam definition och öka förståelsen om konceptet slöseri för att skapa en medvetenhet inom enheten. Dessutom rekommenderas enheten att använda mer omfattande indikatorer så som kumulativt flödesschema för att underlätta identifiering och rotorsaksanalys av slöseri. Slutligen föreslås enheten att eliminera slöseri genom ständiga förbättringar med aktiviteter strukturerade efter PDSA-cykeln.

List of abbreviations and definitions

Expression	Meaning used in thesis
Agile	Flexible and iterative working methods
API	Application programming interface
BE	Business Engine
BI	Business Intelligence
BOD	Business Object Document
IEC	Infor Enterprise Collaborator
IT	Information Technology
Jira	Issue tracking and project management software
LPD	Lean product development
LSD	Lean Software Development
Multiplexing	A method by which multiple signals are combined into one signal over a shared medium. The aim is to share an expensive resource.
Multi-tenant	Customer organizations shares a single running application whilst being isolated with separate sets of data and configurations
On premise	Computer servers locally at customer
POC	Proof of Concept
QA	Quality assurance
Requirements	A condition or capability needed by a user to solve a problem or achieve an objective. Often viewed as the software development equivalent of work-item.
SaaS	Software as a service
Scrum methodology	An agile software development framework, utilized for managing product development in an iterative way.

Single- tenant	One running application per customer
Sprint	Time window of planning in agile software development, typically last from one to four weeks. Work is planned for each sprint where the members are expected to perform only those tasks during the sprint. Sometimes referred to as iteration.
Virtualization	Creation of one or more virtual machines that in turn simulate physical machines that can run software, applications etc.

Table of content

Appendices

1 Introduction

This chapter starts with introducing the background and problem area of this study, followed by presenting the aim of the study. Lastly, the thesis disposition is described, ending the chapter.

1.1 Background

The information technology (IT) industry constantly faces new technological innovations that change how the industry work (Kaltenecker, Hess, & Huesig, 2015). These innovations can either be software or hardware based (Campbell-Kelly, 2001). According to Campbell-Kelly (2001), one example of a technological paradigm shift in the IT-industry occurred between the early '80s to the mid-'90s, when the personal computer commercialized whereby software sales increased by roughly 20 % annually. This caused a power shift within the IT industry where companies like Microsoft, went from minor actors to industry leaders. Following the software providers increasingly dominant role in the IT industry, the internet commercialized in 1995 and expanded rapidly (Campbell-Kelly, 2001). According to Campbell-Kelly (2001), the internet changed how the software industry functions where software no longer required distribution through physical copies, but instead could be delivered to the customer electronically.

As IT is becoming increasingly extensive and complex for organizations and generally considered a noncore competency, the demand for outsourcing such activities to software providers has increased (Demirkan, Cheng & Bandyopadhyay, 2010). Software is traditionally run on computers on the premises of the organizations using the software (Kaltenecker et al., 2015); or on private data centers (Armbrust et al., 2010). However, according to Armbrust et al. (2010), the average server utilization in private data centers range from 5% to 20%, which can be considered wasteful. Organizations also face the risk of underestimating required capacity for peak surges, which can be even more detrimental. According to Demirkan et al. (2010) outsourced hosting of organizations IT-systems has become possible by the innovation of cloud computing which is considered the current technological shift that challenges private data centers and revolutionizes the IT-industry (Náplava, 2016; Saurabh, Young, & Jong, 2016; Dimitrios & Dimitrios, 2012; Li & Li, 2013). Utilizing economies of scale by operating extremely large public data centers at low-cost locations, enables large IT-industry actors like Google and Amazon to offer a “pay as you go” data server service to customers (Armbrust, et al., 2010). According to Armbrust et al. (2010) cloud refers to the hardware and systems software in these public data centers where the service provided generally called utility computing. This makes it possible for software providers to run software on cloud, utilizing utility computing, and in turn deliver applications to the end users over the Internet, generally known as software as a service (SaaS) (Armbrust et al., 2010). According to Armbrust et al. (2010), utility computing together with SaaS is what defines cloud computing and creates a new generic relationship model between utility computing users/provider as well as SaaS users/providers (Armbrust, et al., 2010) and can be seen in Figure 1.

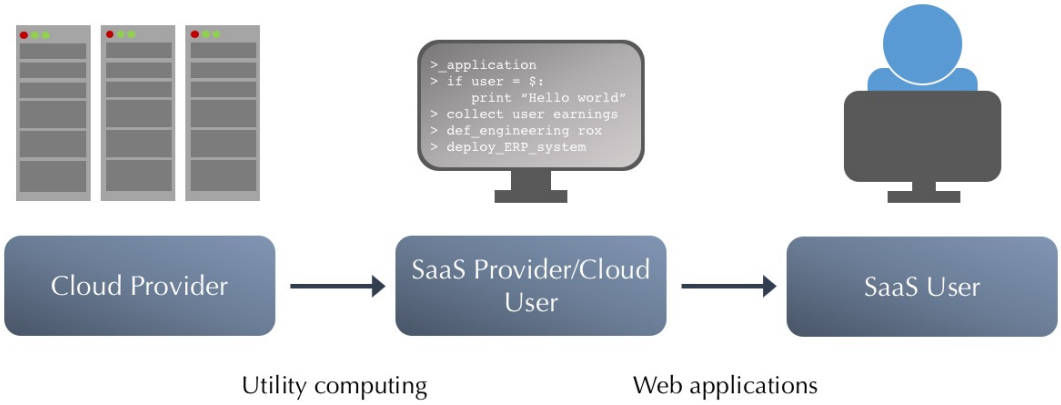

Figure 1: Illustration of the relationship between the cloud provider, SaaS Provider/cloud user, and the SaaS User. The Cloud Provider delivers utility computing to the SaaS Provider/Cloud user who in turn provides web applications to the SaaS user.
Adapted from Armbrust et al. (2010).

Technology like multiplexing[1] and virtualization[2] enables cloud providers increased utilization of their hardware and thus still make a good profit while decreasing financial risk, by eliminating the need of large hardware investment, for other parties in the cloud computing relationship (Armbrust et al., 2010).

Cloud computing introduces a number of benefits for the SaaS users such as increased cost-effective usage of resources as a result of the scalable on-demand service (Saurabh et al., 2016; Li & Li, 2013). SaaS users' capital expenses are converted to operational expenses without the risk of running out of capacity or overinvesting in servers (Armbrust et al., 2010). Additionally, cloud computing enables small and medium-sized enterprises (SMEs), that earlier could not afford the investment of software licenses and hardware, to exploit the benefits of IT-systems in their business and be charged on an ongoing basis (Demirkan et al., 2010; Kaltenecker et al., 2015).

From the perspective of software providers, cloud computing offers more benefits beyond an increased market of SMEs as customers. Some of which is liberation of IT infrastructure set-up, more efficient deployment of software, increased addressability and traceability (Goutas, Sutanto, & Aldarbesti, 2016). SaaS challenges the traditional software licensing model in favor for a subscription based approach (Armbrust, o.a., 2010) where SaaS providers derive their profits from the margin between the cost of utility computing and the revenue generated from customers subscriptions (Li & Li, 2013). Hence, cost-efficient use of the cloud providers' services is vital for SaaS providers in order to maximize revenue. The most efficient utilization of the cloud provider's infrastructure is by the feature of multi-tenancy in which multiple tenants (customer organization) shares a single running application whilst being isolated with separate sets of data and configurations (Samrajesh, Gopalan, & Suresh, 2016; Li & Li, 2013). Since multiple customers can share the same application and

[1] A method by which multiple signals are combined into one signal over a shared medium. The aim is to share an expensive resource. (https://en.wikipedia.org/wiki/Multiplexing)

[2] Creation of one or more virtual machines that in turn simulate physical machines that can run software, applications etc. (Dimitrios & Dimitrios, 2012; Naone, 2009)

infrastructure the SaaS provider can lower operational costs of utility computing and truly reap benefits from economies of scale available in the cloud computing relationship (Chou & Chiang, 2013). Moreover, cloud computing utilizing multi-tenancy applications introduces several additional benefits for the SaaS provider due to the single software version serving multiple customers including reduced software development time, centralized version control and lower maintenance cost (Samrajesh et al., 2016).

Cloud computing appears to offer significant benefits to all members of the value chain that support the notion of a major technological paradigm shift. However, research indicates that new challenges are introduced to software providers when transitioning to a SaaS model. Vidyanand (2007) means that SaaS gives customers more bargaining power since the customer no longer needs to invest in the technology needed for the operation of the system and thus not tethered to the hardware in the same way. Chou & Chiang (2013) as well as Goode, Lin, Tsai & Jiang (2015) supports this and further mentions that the unique features of SaaS results in lower switching costs and hence emphasizes customer satisfaction as the key role to avoid clients switching to new vendors. Kim, Hong, Min & Lee (2011) means that high customer retention rates are increasingly essential for the software providers longevity when operating a SaaS model since acquiring a new customer cost more compared to retaining an existing customer. According to Goode et al. (2015), compliance with clients' operational requirements is the most significant contributor to customer satisfaction. Moreover, Goode et al. (2015) emphasize rapid fulfillment of customer expectations is vital and seen as a minimum requirement in a SaaS relationship due to the nature of the service model. However, SaaS makes software providers fully responsible for the maintenance of the software where customers, or third-party consultants, are disabled from making own modifications on the software which in turn implies that all customer requirements must be met by the SaaS provider (Kaltenecker et al., 2015).

Beyond fulfillment of operational requirements, in order to retain their customer base, software providers must deliver high service quality. Research shows that the most important attributes of SaaS, in order to maintain a high perceived service quality and in turn satisfied customers, are flexibility of contractual and technical[3] changes to the service (Kim et al., 2011; Chou & Chiang, 2013) as well as being responsive (Chou & Chiang, 2013; Goode et. al. 2015) and able to offer customizable services (Goutas et al., 2016). These attributes, flexibility and responsiveness and customization, is shown to be more important in a SaaS model compared to a license-based software service. To summarize, SaaS gives the software provider an increased amount of potential clients and easier management of further development due to single version software but results in full responsibility of delivering a responsive service and maintenance with flexibility in a customizable product that satisfies all clients technical and operational requirements. In order to succeed in these aspects and deliver high service quality, the software provider needs to focus development resources on fewer but more effective innovations that are likely to deliver satisfying outcomes (Goode et al., 2015) and deliver these in reliable and frequent upgrades (Chou & Chiang, 2013). This calls for increased productivity, in order to fulfill an increased amount of customers' expectations, but also a shortened lead time, in order to be perceived as responsive.

The increased pressure on software providers in a cloud computing relationship requires an efficient development of software. Poppendieck & Poppendieck (2003) exemplifies the

[3] E.g. functionality, scalability, interoperability, or modularity of the application

diversity of productivity between software development organizations by referring to a system developed separately by both Florida and Minnesota State. Florida spent 15 years and $230 million developing roughly the same system as Minnesota completed in a year at the cost of $1.1 million. This 200:1 difference in productivity demonstrates the need for efficient approaches to software development. Most development approaches origins from the waterfall model developed in the ‘70s (Stoica, Mircea, Uscatu, & Ghilic-Micu, 2016) and follows a sequential and linear process with comprehensive front-end planning (Marcello, 2016). This model has dominated the software industry since the early '70s until late ‘90s and is used even to this date, even though the model has proven inflexible. In 2001, the agile manifesto was introduced in Utah, the USA, after a meeting between representatives from software development organizations, where a more iterative software development methodology was developed (Stoica, et al., 2016). Agile methods have since been a commonly used in software development and gained an increased popularity in the past decade (Sulaiman, MohdNaz’ri, & RasimahCheMohd, 2016). However, even agile methods have been argued to be insufficient in many software development organizations whereby the IT-industry have gazed to other industries for better answers.

Lean is a concept that origins from manufacturing, while the fundamental idea of lean is applicable to other areas (Poppendieck & Poppendieck, 2003). Middleton & Joyce (2012) mentions that lean is a concept of reducing lead time compared to agile methods where the fundamental idea is to plan less concrete. Moreover, the effect of lean has been shown to double the productivity in both manufacturing and service organizations (Middelton & Joyce, 2012) whereby the interest of lean in software development has increased. Poppendieck & Poppendieck (2003), introduced an adapted concept of lean to a software development context i.e. Lean Software Development (LSD) and thus simplified the adoption of lean in software development organizations. Agile practitioners have an increasing interest in LSD as a complement to agile methods, while others even claim that LSD is the next disruptive innovation in software processes (Wang, Conboy, & Cawley, 2012). However, LSD is still seen as a fairly new concept but has shown to be an effective and profitable way to manage software development (Middelton & Joyce, 2012; Rodríguez, Partanen, Kuvaja, & Oivo, 2014).

One of the most fundamental principles of Lean is the elimination of waste. Poppendieck & Poppendieck (2003) defines waste as anything that does not add value to the product perceived by the customer. Hence, waste can be identified as everything that gets in the way of quickly satisfying the customer needs, something that is increasingly important for SaaS providers. Al-Baik & Miller (2014) have shown that by utilizing the concept of waste elimination in lean, lead-time was reduced by over 55% for a software provider in a case study. Moreover, the authors state that waste should be eliminated by continuous and incremental improvements. A company that faces these challenges is Infor M3.

1.2 Problem discussion - Infor M3 - Integration & BI unit

One company that has recognized the possibilities of multi-tenant SaaS solutions is Infor, one of the world’s leading enterprise software providers. Infor is a company consisting of multiple enterprise software acquisitions, one of which is the Enterprise Resource Planning (ERP) system Infor M3 (more information about Infor and Infor M3 can be found in Section 4.1 and 4.2). Infor M3 have spent the last couple of years transforming their product to a multi-tenant SaaS solution, earlier only providing on premise and single tenant solutions to its customers. When providing a SaaS service, Infor M3 suddenly becomes responsible for maintenance, upgrades and minimizing downtime of the software, activities that earlier have been done at IT- departments at the customer. This sets entirely new demands regarding

responsiveness and flexibility on the employees, products, and process of Infor M3 which in turn requires increased flow and short lead-times in their development. One unit of Infor M3, called Integration & BI (Business Intelligence), have discovered shortcomings in their current working methodologies in order to comply with the increased demands a multi-tenant SaaS solution that is required of the organization. Henceforth, the manager of the unit has developed and introduced several principles similar to the principles of lean software development, where one of the concordant principles between the manager's and lean principles is to *Eliminate waste*.

In order for Infor M3 – Integration & BI to successfully compete in a SaaS environment, the unit states: "We have to be able to deliver flawless products as often as possible", which is challenging to accomplish with the presence of waste in their development process. However, the unit was still in an early stage of implementing the principles when this thesis was conducted and the general knowledge level regarding how to understand, identify and eliminate waste was low. Consequently, Integration & BI unit is a valid candidate for a case study in order to research implementation of the lean software development principle *Eliminate waste*. The expected synergy between the case company and research together with the support from the unit's manager regarding the *Eliminate waste* principle will increase the likelihood of successful cooperation.

1.3 Aim

The aim of this thesis is to identify waste and suggest approaches to reduce waste at the unit Integration & BI at Infor M3. In order to accomplish this, three study questions have been formulated. The purpose of the first question is to create an understanding considering waste in software development, based on earlier research.

SQ1: What is considered waste in software development?

In order to reach Integration & BI's goal, successful waste reduction approaches must be developed. However, in order to accomplish this, it is fundamental for Integration & BI to create an understanding about waste and how it functions, based on the unit's novelty regarding the subject. Furthermore, identification of occurring wastes has to be done, as well as classification of the most prominent waste with corresponding root causes in order to find the most suitable improvement efforts for the unit. This may be done by utilizing findings from SQ1 as a foundation in order for Integration & BI to understand their current situation regarding waste.

SQ2: What is Integration & BI's current waste situation?

With the perception of both state-of-the-art literature and Integration & BI, potential improvements of approaches can be identified. The following study question aims to synthesize current practice and empirical findings in order to further suggest waste reduction approaches at Integration & BI.

SQ3: How can Integration & BI reduce waste in their software development?

1.4 Thesis disposition

The disposition of this thesis is made out of five different parts excluding the introduction, all of which are made in order to answer the previously stated questions and thereby fulfilling the aim of the study. Firstly, the various research methods chosen for the study are clarified and justified by the authors. Followed by a theoretical frame of references, and literature review, covering previous academic research within the field. This is then followed by the

company description, containing a situation analysis of the investigated teams at Integration & BI, as well as a short description of Infor and Infor M3. Further, the Waste Identification and Analysis cover the collected information from the conducted interviews, based on facts and thoughts regarding waste within the teams. In order to simplify this for the reader, the chapter will be a mix of both empirical findings as well as analysis of the different team's current situations. Moreover, an analysis is conducted in Waste elimination approaches at Integration & BI, comparing the findings from Integration & BI, with the previously gathered research in the literature review. Then this culminate into the conclusions and recommendations of the study, which present the most important findings from the thesis and gives recommendations to the case company Integration & BI. The relationship between the study questions and the different parts of the thesis is presented in Figure 2.

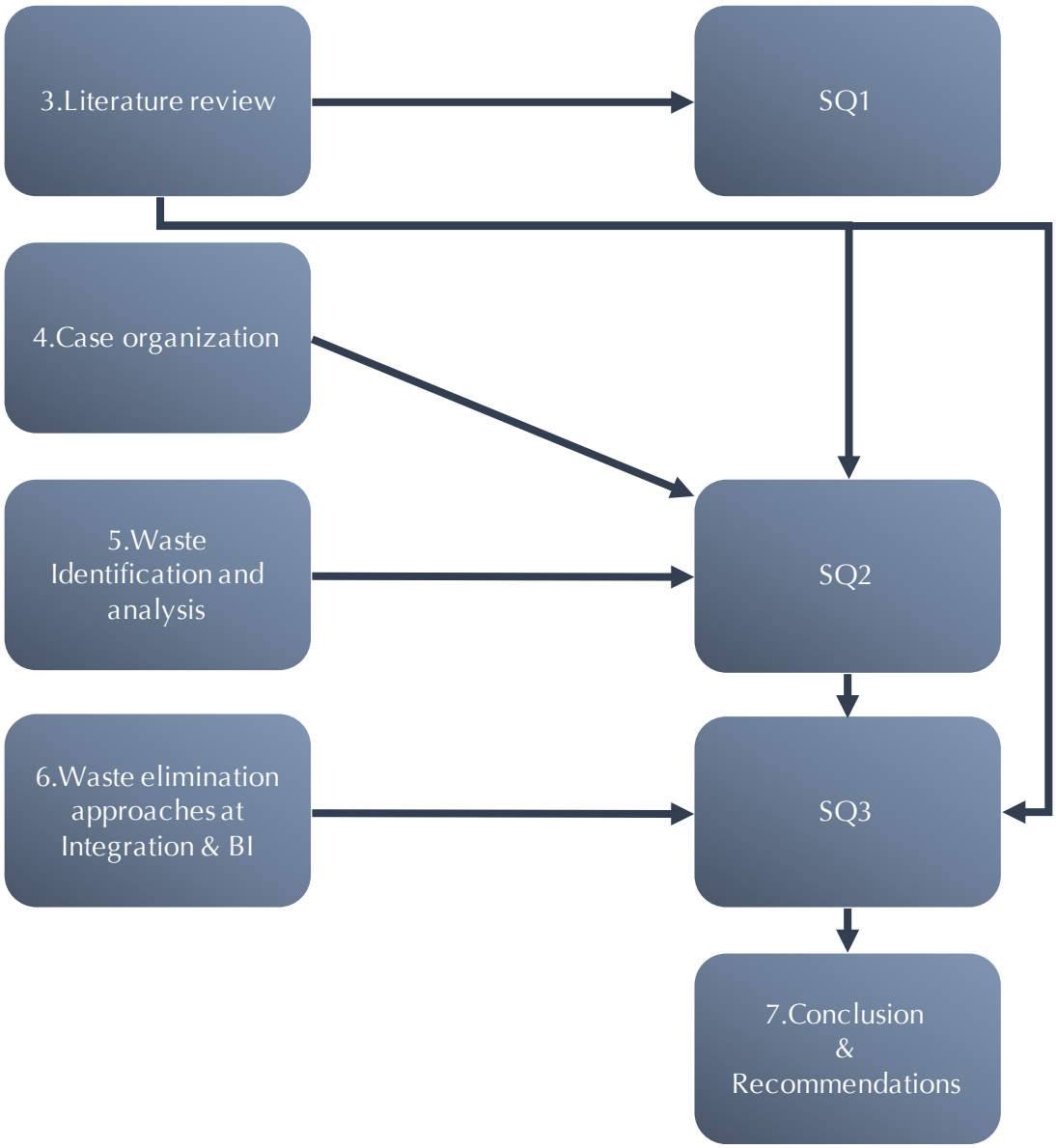

Figure 2: Relationship between the study questions and the different parts of the thesis

2 Method

This chapter describes the method chosen in order to fulfill the aim of the study. Table 1 is a summary of the various methodology choices made in this study. Each methodology will be additionally explained in its own section together with reasoning regarding the method choice.

In order to answer the study questions and achieve the aim of the study, a specific methodology was followed. Below in Table 1, a summary of the chosen methods during this study is presented.

Table 1: Summary of the chosen methods

Methodology	Chosen Methods
Research Purpose	Descriptive, Explanatory
Research Approach	Deductive
Qualitative & Quantitative methods	Qualitative
Research Strategy	Case study
Data collection	Interviews, Observations
Sample of respondents	Non-probability sample
Data analysis	Pattern matching
Critical review of methodology	Triangulation, Participant validation

2.1 Research purpose

Saunders Lewis, & Thornhill (2016) state that the purpose of research will be either exploratory, descriptive, explanatory, evaluative or a combination of these depending on the design of the study questions, which David & Sutton (2016) concurs with. Exploratory studies are useful when the researcher desires to elucidate the understanding of an issue, often with a very small amount of previous expositions (David & Sutton, 2016; Björklund & Paulsson, 2012). Saunders et al. (2016) argue that exploratory research is flexible and adaptable to changes, something a researcher conducting an exploratory study has to be, as the aim of the exploratory research is to reveal knowledge about unexplored areas. The purpose of descriptive research is according to Saunders et al. (2016) to obtain precise descriptions of individuals or occurrences. Furthermore, Saunders et al. (2016) state that descriptive research often lays the foundation for explanatory research and the combination of these research purposes are called descripto-explanatory studies. On the other hand, pure explanatory studies focus on finding relationships between variables and the given effects of these relationships (Saunders et al., 2016; David & Sutton, 2016). Explanatory studies can also be conducted when researchers search "for deeper knowledge and understanding of a subject, that needs to be described and explained" (Björklund & Paulsson, page 60, 2012). Lastly, evaluative studies purpose is to discover how well something actually works.

Evaluative studies do not only contribute with facts concerning ”how” well something is working but also ”why” it is working well (Saunders et al., 2016).

In this study, the combination of descriptive and explanatory reasoning, namely descripto-explanatory, has been adopted to answer the three study questions. The first two study questions, SQ1 and SQ2 are more descriptive in nature and creating the foundation of knowledge which the last study question, SQ3, requires in order to be answered. SQ3 are considered as the explanatory part of the study where relationships between different variables are being examined in order to identify the most suitable recommendations.

2.2 Research approach

Björklund & Paulsson (2012, p. 64) argues that during a study “researchers are moving between different levels of abstraction, where theory and empirical studies constitute the endpoints.”. The design of a study often comes down to three different approaches deductive, inductive or abductive, according to Saunders et al. (2016), which is also supported by Björklund & Paulsson (2012).

David & Sutton (2016, p.83) argues that deductive research is when ”researchers try to test and prove a hypothesis”, while Björklund & Paulsson (2012), as well as Saunders et al. (2016), means that the deductive approach starts from different theories and these theories then constitutes the foundation of on which assumptions will be made upon, regarding the coming empirical findings. The researcher then tries to verify these predictions when analyzing the collected data from the study (Björklund & Paulsson, 2012). Inductive research is about exploring a field according to David & Sutton (2016) while Björklund & Paulsson (2012) further explains inductive research as making assumptions from empirical studies. Thus, instead of using previous research, the researcher develops new theory from empirical studies in order to create frameworks (Saunders et al., 2016). Moreover, if elements from both deductive and inductive approaches are used and the study goes back and forth between these two, the approach is called abductive (Björklund & Paulsson, 2012; Saunders et al., 2016).

In order to answer the study questions, the authors have used a deductive approach. By using theory as a foundation regarding waste within software development, the authors could assume that at least some of the waste classifications by Poppendieck and Poppendieck (2006), would be relevant at Integration & BI. Nevertheless, inductive elements as interview results can be found in the thesis. However, the authors choose to classify the study as deductive, since the inductive elements are minor in comparison to deductive elements.

2.3 Qualitative and quantitative methods

Information can be collected either by a qualitative or quantitative method (Saunders et al., 2016; Björklund & Paulsson, 2012; & David & Sutton, 2016). Quantitative methods are used when the data is numerical and therefore often linked to the usage of surveys and statistical investigations (Saunders et al. 2016). Qualitative methods are used when the researcher seek to create a deeper understanding regarding a certain problem (Björklund & Paulsson, 2012). The information gathered during a qualitative study is often expressed in words according to David & Sutton (2016) where common instruments used are observations and interviews (Saunders et al. 2016). However, Saunders et al. (2016) argue that a study often combines elements from both quantitative and qualitative methods.

The majority of the data collected in this study was expressed in words, collected from interviews and observations. Hence, it can be concluded that this study is of qualitative

nature. Moreover, a qualitative method also matches the purpose of the study where creating a deeper knowledge about waste identification and elimination is the primary purpose. Since the area is complex, the authors decided that a survey or questionnaire would not have been adequate in terms of answering the sought questions. Furthermore, there was no known stored data regarding waste at Integration & BI when the study was conducted, which further reduce the rationality behind a quantitative method for this study.

2.4 Research strategy

A research strategy is according to Saunders et al. (2016) a strategy chosen to answer the study questions. When conducting a qualitative research, one of the most common research strategies is the case study (Saunders et al., 2016). A case study is a profound study of a unit which can be everything from an individual to a multinational organization (David & Sutton, 2016; Saunders et al., 2016). David & Sutton (2016) state that case studies use methods such as interviews, focus groups and observations to collect the desired data.

Yin (2009) state that there are four design choices for case studies: single-case holistic, single-case embedded, multi-case holistic and multi-case embedded. The single case study is mostly used when the case represents an environment where existing theory can be tested or when the case offer uncommon or unique conditions (Yin, 2009). Multiple-cases are often done so the researcher can investigate if their findings are possible to replicate to other circumstances (Saunders et al. 2016). If a case study is holistic there is only one entity that is being researched, while during an embedded case study, there can be more than one area of focus (Yin, 2009).

The performed study has been done at Integration & BI, which was the only company investigated in the study, making it a single-case study. Furthermore, the study has been of a qualitative nature which strengthens the choice of a case study. During data collection and analysis of waste at Integration & BI, several teams were investigated since the different teams had various conditions. The authors could thereby identify various explanations to the waste that existed within the different teams. The study can consequently be seen as an embedded single-case study.

2.5 Data collection

In order to answer the study questions and reach the aim of the study different kinds of data was collected, where the data was either of primary or secondary nature. The methods utilized to gather this information will be presented in this section.

2.5.1 Primary data

According to Saunders et al. (2009), primary data is information collected during the study that the researcher is gathering by themselves. There are a number of ways of collecting primary data, for example, observations, interviews, surveys and experiments (Saunders et al., 2009). In this study interviews and observations was conducted as methods to gather primary data and therefore other methods were excluded from the report.

2.5.2 Interview

An interview involves asking questions and listening to the given answers by the respondent according to David & Sutton (2016). Interviews are mostly done in person, but can also be made using telephones or different types of computer communication programs (David & Sutton, 2016). The main purpose of the usage of interviews is to collect desired data from the respondent, that later can be utilized to answer the study questions (Saunders et al., 2009).

Interviews can be designed with regards to two dimensions; structured or unstructured interviews and standardized or non-standardized interviews (David & Sutton, 2016). The purpose of a structured interview is to preserve the questions asked from one interview to another and thereby maintain the repeatability and reliability, while unstructured interview seeks after deep validity and therefore the interviewer ask questions that make the respondent responsible for the flow of the interview (David & Sutton, 2016). Standardized format of an interview intends to ask questions that promotes closed answers, which are easier to quantify compared to the more open questions asked during an unstandardized interview which instead gives more detailed and developed answers (David & Sutton, 2016).

Interviews are commonly used in case studies and this study is no exception. First, a pilot interview was conducted to ensure that the questions and structure of the interview were functioning. To confirm that the answers given by the interviewees were reliable and valid the interviewees were provided with a summary regarding the subject before the interview, explaining waste in software development and thereby gave the respondent time to reflect on the subject. The summary together with the interview guide were both based on the seven wastes by Poppendieck & Poppendieck (2006) which was considered the most legitimate by the authors. The summary consisted of Section 3.2.1. Moreover, the supervisor at Integration & BI was familiar with these and verified the applicability of the classifications on the unit. Thus, this made Poppendieck & Poppendieck (2006) waste classification the obvious choice over other waste classification models.

During the pilot, it became clear that a semi-structured with non-standardized questions was the right way to go. The differences between the teams and the complexity of the topic made it unfeasible to conduct interviews with standardized questions. Furthermore, semi-structured interviews were chosen since the authors tried to maintain the repeatability between the interviews, while also gather answers of which create a deeper knowledge regarding the area. Poppendieck & Poppendieck (2006) also state that the categorization of waste is not meant as a tool to classify waste, but instead as a thinking tool enabling individuals to recognize and identify why certain behaviors, processes, etc. are wasteful activities. Consequently, identification of waste in work environments is a cognitive process that requires discussion in order to recognize the nature of the waste. Subsequently, unstructured and non-standardized interviews were most suitable since the goal was primarily to identify and understand the waste.

The individuals that were interviewed originated from different backgrounds and had different roles, which is discussed further in section 3.6. The teams interviewed was spread across several countries, hence, the interviews were conducted in various ways. Only one interview was done face to face at the office in Stockholm, while the rest of the interviews were done using Skype. All of the interviews were about one hour except one that lasted for almost two hours with a Swedish employee. Five interviews were conducted in total, where one interview was performed with three people, two with two people and lastly two interviews with one individual. Why the number of interviewees varied between the teams, was due to the fact that some teams were larger and in order to identify all of the wastes, more people had to be interviewed. In Appendix C, the interview guide that was used as guidance and support during the different interviews can be found.

2.5.3 Observations

Observations involve recording, observing and analyzing the behavior of an entity, and is a great way to understand an ongoing situation (Saunders et al., 2009). There are two kinds of observations: participant observations and structured observations (Saunders et al., 2009). Participant observations are a qualitative method and comprise that the researcher tries to

participate in the daily life of its subjects (Saunders et al., 2009). According to Saunders et al. (2009), this does not only enable the researcher to observe but also be a part of the organization or community. The structured observation is a quantitative method and involves observations whose purpose is to show how often things happened, rather than why they happened, which can translate into quantifying a behavior (Saunders et al., 2009).

In this study, participant observations were used to understand different activities and operations at Infor M3. In order to get an understanding of the organization, participation and observations were conducted to obtain the desired information. Since the authors were not used to the way of work within the software development industry and the complexity of the product, a lot of information was needed to be able to build an understanding of the business and the processes involved. Participant observations were conducted through meetings at the office, at lunches and meetings through Skype when the Philippine team participated. These participations were mainly done during the visit at the office in Stockholm, that lasted from the end of January until mid-February, while Skype meetings were conducted primarily before the visit in Stockholm and when meetings with the teams in the Philippines occurred. Furthermore, weekly meetings with the supervisor from Integration & BI was conducted using Skype.

2.5.4 Secondary data

Saunders et al. (2016) and Björklund & Paulsson (2012) state that secondary data is information that was collected for one purpose, however, utilized with a different purpose by other researchers. Secondary data can be limited in terms of availability and the level of quality of the material can vary (David & Sutton, 2016). Since the chosen area of this study still is in an early stage, different mediums have been used to find the desired data. Books, journals, articles and conference notes have been investigated to obtain desired information. Various search engines have been used: Google Scholar, the university library at Luleå University of Technology and Scopus. Different key words were used to attain the desired information, all of which are presented below.

Key words: Lean software development, IT, Software Industry, on premise, SaaS, Cloud computing, Waste, Waste management, IT/Software Industry.

2.6 Selection of respondents

A population can be considered as every entity that the researcher want to include in a study (David & Sutton, 2016). However, David & Sutton (2016) and Saunders et al. (2016) state that every unit in a population should be considered if possible, but in cases where the population is considered too big, samples that can represent the whole population has to be used. According to both Saunders et al. (2016) and David & Sutton (2016), there are two major categories of sampling techniques: probability sampling and non-probability sampling. If the probability to choose a case is the same for all the samples of the population, probability sampling is used (David & Sutton, 2016; Saunders et al., 2016). Further, Saunders, et al. (2016) argue that probability sampling is used when the researcher needs to statistically estimate the features of the target population in order to fulfill the aim of the study. Thus, probability sampling is often linked to the usage of surveys and experiment research strategies (Saunders et al., 2016). The second sampling technique, non-probability sampling is used when all of the possible cases in a population are hard to identify, or when time or cost restrictions makes probability sampling unpractical (David & Sutton, 2016).

In this thesis, a non-probability sampling has been used, primarily because of the time limitations during the study. Moreover, the selections of the samples were not conducted

randomly, making non-probability sampling as the only viable choice (Saunders et al., 2016). Furthermore, there are four different types of non-probability sampling according to Saunders et al. (2016), which are: quota, purposive, volunteer and haphazard sampling. The sampling method used in this thesis is the purposive sampling method and therefore the only one further discussed in the thesis. The purposive method grants the researcher possibility to choose certain individuals that are believed to be suitable for the research area (David & Sutton, 2016). The purposive method is often called judgmental sampling and is used when the samples are small, like in case studies (Saunders et al., 2016).

In order to fulfill the aim of the study, the most suitable individuals at Integration & BI had to be chosen for interviews. Since the different teams are spread globally, making it difficult to know who is the most suitable, the supervisor at the unit helped the authors with this selection. The supervisor holds a higher level position, making his choice of respondents legitimate and thereby a trusted judge according to the authors. Among the teams at Integration & BI, employees are spread across Sweden, Germany, Philippines and USA, however, the main part of the employees are located in Sweden and Philippines and only a handful of employees are located at the other sites. Development was primarily done in the Philippines, whilst the teams in Sweden had more of a coordination and project management focus. Hence, the selected individuals were both from the Swedish and Philippine teams, where the roles and number of interviewees are presented in Table 2.

Table 2: Summary of the selected respondents

Team	Interviewees	Country	Position
IEC	3	Philippines	Manager IEC, Principle software engineers (PSE)
Integration	1	Sweden	Principle business analyst (PBA)
BI	1	Philippines	Principle software engineer (PSE)
BI	1	Sweden	Senior product manager (SPM)
BOD	2	Philippines	Manager Integration/M3A, Senior software engineer (SSE)
BOD	1	Sweden	Principle business analyst (PBA)

Poppendieck & Poppendieck (2006) state that waste differs from one case to another and thereby it was important that all of these teams with various conditions were interviewed, in order to ensure the classification of waste at the majority of the teams at Integration & BI.

2.7 Data analysis

Analyzing qualitative data is about transforming all of the collected data from interviews and observations into a more manageable amount (Adams, Khan, & Raeside, 2014). Furthermore, Adams et al. (2014) determine that data must be prepared at the start of the analysis and generally all the data cannot be stored. However, the researcher should try to keep as much of the information as possible. Interviews are often audio-recorded, like in this study, and one way to capture this data is to transcribe it. Transcription means that the researcher reproduces the spoken words from the interviews into written words, verbatim (Saunders et al., 2016).

Furthermore, Saunders et al. (2016) discuss several different methods that aid the analysis, where one method is to make a transcript summary. By doing this the researchers can compress larger fragments of the text, but still maintain the vigorous of the text (Saunders et al., 2016).

Yin (2009) presents five different analysis techniques: Pattern matching, Explanation Building, Time Series Analysis, Logic Models and Cross-Case Synthesis. In this thesis, pattern matching has been used. Pattern matching is about comparing predicted patterns with patterns found during the data collection (Yin, 2009). Yin (2009) further state that if these patterns collide, the internal validity of the study strengthens.

In order to retain the desired information from the conducted interviews, the interviews were transcribed. After the transcription, each interview was summarized into a more tangible amount of information. To be able to treat each interview in a similar way and by that simplify the analysis phase, a table were developed. This table is shown in Appendix A and was used so the authors could use a standardized method while summarizing the different interviews and therefore save time. The gathered data was afterward compared with the findings from the literature review. Further, the findings from the different interviews and observations were compared to each other, to investigate if there was a common pattern between the different teams and the occurring wastes.

2.8 Critical review of the research methodology

Researchers always strive to achieve qualitative research that others will see as reliable (Saunders et al., 2016). Two different dimensions that measure the credibility of a study is reliability and validity (Björklund & Paulsson, 2012). Saunders et al. (2016) argue that credibility of a study may be promoted if the interviewers in beforehand provide the interviewee with information concerning the areas that will be discussed during the interview. This will help the interviewee to be prepared for the interview, hence the answers and information gathered during the interview will be more legitimate, which in turn strengthen the validity and reliability of the study (Saunders et al., 2016).

Björklund & Paulsson (2012) argues that validity can be considered as the extent a researcher actually measures what is supposed to be measured, while reliability is concerning how reliable the measurements during the study have been. Thus, reliability refers to the ability to reconstruct a result multiple times (Björklund & Paulsson, 2012; Saunders et al., 2016). Furthermore, David & Sutton (2016) state that there are two kinds of validity, internal and external, which Saunders et al. (2016) concur with. Internal validity refers to the relationships within the actual data being studied and is established when these relationships are demonstrated (Saunders et al., 2016; David & Sutton, 2016). External validity, also known as generalizability, aims to investigate if the study conducted can be applicable to other entities from the population, which the chosen respondents originate from, or if its applicable to other relevant settings or groups (David & Sutton, 2016; Saunders et al., 2016). Yin (2009) mentions that according to critics, single case studies often provide poor generalization and thereby poor external validity, and has been concerned as one of the biggest problems with the execution of a case study. However, Yin (2009) argue that many of these critics think of statistical generalization regarding case studies, while Yin (2009) would like to think of analytical generalization. According to Yin (2009), analytical generalization concern's the generalization of specific results against a theory, creating higher external validity.

In order to increase validity and reliability, different tools can be used. Saunders et al. (2016) present two different tools, triangulation, and participant validation. By using triangulation, the researchers do not rely on a single source, but instead conducting a study from a

collection of different sources to strengthen the validity and reliability. The second tool participant validation regards the minimization of misunderstandings between the researcher and respondents (Saunders et al., 2016). This is often used when interviews or observations have been conducted and refers to the re-sending of information to the respondent to assure the accuracy of the information (Saunders et al., 2016).

During the study both triangulation and participant, validation has been used. In the literature review and also in the case of interviews, several sources have been used to assure validity within the study. Regarding the participant validation, information concerning the case organization and associated observations have been sent to the supervisor at Integration & BI for validation. Further, the interviews were audio recorded to reassure that no information was lost through the enablement of re-listening which not only strengthen validity, but also the reliability of the study. However, the recorded audio files were not sent back to the interviewees, which could have been done to further strengthen the validity. Nevertheless, the interviewees were sent information before the interview preparing them for the occasion and with simple interview questions, room for inaccuracy was minimized. Moreover, the authors conducted a draft presentation with all of the previously involved interviewees where findings of the study were presented. During the presentation, the attendants had the possibility to ask questions and correct the authors regarding things that was misinterpreted by the authors. Thus ensured a strengthen validity of the study.

The report was sent to both the supervisor at Luleå University of Technology and opponents to ensure the absence of errors. Lastly, the external validity of the study is difficult to ensure, since the study has been conducted on a single case company. However, the results from the study can be applicable against theories regarding the software development area, making the external validity to increase. Further Poppendieck and Poppendieck (2006) argues that the different wastes presented in section 3.2.1, also often are very dissimilar between different practitioners, making it difficult for potential recommendations to be adopted on a general basis.

3 Literature review

This chapter will cover the theoretical foundation, of which the analysis and recommendations are based on. Moreover, the literature review will serve as a basis for the material used to create the empirical study. Lastly, the material presented in this chapter will help the reader to comprehend the significant subjects that were investigated in this study.

3.1 Lean

The concept lean originates from Toyotas manufacturing strategy at their production system: Toyota Production system (TPS) (Liker, 2009). The lean concept is generally seen as a philosophy (Antosz & Stadnicka, 2017; Bhasin & Burcher, 2006; Wallstrom & Chroneer, 2016); where Liker (2004) defines lean as: "A philosophy that when implemented reduces the time from customer order to delivery by eliminating sources of waste in the production flow" (p. 481).

According to Womack, Jones, & Roos (1990) the difference between traditional working methods and lean production, is the endeavor of perfection, meaning that continuous work regarding minimizing *Defects*, costs and inventory are carried out endlessly. In order to concretize this philosophy, Liker (2009) identified and documented 14 principles that work as the basis for TPS (Liker, 2009). In addition to the 14 principles, Liker (2009) identified seven Muda's, also known as waste during his study which of today is a central concept of the lean philosophy.

Liker (2009) mentions that waste is defined by Toyota as everything that consumes time, but does not contribute any value to the customer. Ohno (1988) highlights the importance of waste elimination and argues that improving efficiency only makes sense when it is linked to cost reduction. However, Wallstrom & Chroneer (2016) disagrees and claims that lean covers waste from resources in general, and not only cost related waste that TPS mostly focused on. Furthermore, the founder of TPS, Ohno (1988) is according to Womack & Jones (2003) the first to describe and classify the Seven Wastes as *Transport*, *Inventory*, *Motion*, *Waiting*, *Overproduction*, *Extra processing*, and *Defects*. This perspective has later become the general classification of wastes. However, often with small contributions or changes by later researchers such as Liker (2004); Bhasin & Burcher (2006); & Womack & Jones (2003). Furthermore, Liker (2009) suggests that the wastes are not only applicable in production, but also in product development and administration. Womack, Jones, & Roos (1990) agrees with the suggestion of Liker (2009), but further argues that lean philosophy is relevant in any industry.

3.2 Lean Software Development

The relevance of lean philosophy in any industry is supported by the increasing amount of research regarding applying lean principles in software development. However, according to Wang, Conboy & Cawley (2012) as well as Khurum, Petersen & Gorschek (2014), the present understanding of lean software development is largely driven by practitioners' writings such as Poppendieck & Poppendieck (2003). Hibbs, Jawett & Sullivan (2009) mean that most subsequent work regarding the topic of lean software development is based on Poppendieck & Poppendieck (2003) and the lean principles they have identified suitable when operating in a software development context. However, the idea of applying lean to software development dates back to research by Freeman (1992) while the concept with a full array of principles can be considered founded by Poppendieck & Poppendieck (2003).

Lean principles are well documented, yet an inconsistency in success when applying them is observed by Poppendieck & Poppendieck (2003) who argues that the nature of the result stems from organizations capability to change the culture and organizational habits. Moreover, organizations that have captured the essence of lean thinking have realized significant performance increases. They continue with this argument by stating that principles are universal guiding ideas while practices give guidance in what you do, but needs to be adapted to the context. Consequently, Poppendieck & Poppendieck (2003) means that there is no such thing as best practice transferable from one organization to another. However, this is rarely taken into account when applying metaphors from other disciplines into a software development domain and the inevitable result is unsatisfactory. Poppendieck & Poppendieck (2003) describes this as transferring of practices rather than principles and should be avoided. Table 3 presents the seven waste classifications developed by Poppendieck & Poppendieck (2006).

Table 3: : Lean principles by Poppendieck & Poppendieck (2006)

Lean software development Principle	Description
Eliminate waste	In order to recognize waste, developing a sense of value to be able to recognize actions that do not add value. When this is known, it is possible to identify waste and later eliminate it.
Build quality in	Instead of only create qualitative when testing, build quality into the code from the start. Thereby the creation of defects is limited in advance.
Create knowledge	When generating knowledge, it should be codified and implemented into the organizational knowledge base. Thereby accessible to the rest of the organization.
Defer commitment	Decisions should as often as possible be reversible and easy to change. However, if the decision is irreversible, it should be decided as late as possible.
Deliver fast	Software should be delivered at such a fast rate, that customers do not have the time to change their minds. Moreover, enable later decision making.
Respect People	The respect in this principle refers to the ability to hand out responsibility. In order for an employee to thrive and flourish, it needs to feel respected by the organization.
Optimize the whole	Optimizing should only be conducted on the entire value stream since sub-optimization often result in decreased flow.

Petersen & Wholin (2011) means that lean software development distinguishes from other modern software development approaches in the end to end focus of the value flow and the unique perspective of waste. Thus, facilitating software development organizations to more successfully improving their software development processes (Petersen & Wohlin, 2010).

3.2.1 Eliminate waste

Elimination of waste can be seen as a core activity for organizations pursuing the lean philosophy, which is also true in lean software development. The definition of waste in lean software development is coherent with Toyota's perspective where Poppendieck & Poppendieck (2003) defines waste as "…anything that does not add value to a product, value as perceived by the customer" (p. xxv). The customer focus is an important element and should be included in any proper definition of waste, such as Rodríguez, Partanen, Kuvaja, & Oivo's (2014): "Everything done in the organization should produce value to the customer. Thus, if something absorbs resources but produces no value, it is considered waste and has to be removed" (p. 4771). Consequently, the ultimate situation is when organizations know exactly what their customers want and deliver exactly that, virtually immediately, pursuant to Poppendieck & Poppendieck (2003). Hence, waste can be described as anything that gets in the way of quickly satisfying the customer needs. According to Poppendieck & Poppendieck (2003) comprehending the concept of waste can be a high hurdle for organizations as it can seem counterintuitive initially when bureaucratically ways of thinking and practices are deeply rooted in the organizational culture and thus difficult to change.

Poppendieck & Poppendieck (2003) have translated Toyota's seven categories of waste in manufacturing to a software development domain. However, the novelty of the topic can be indicated since the categories of waste were updated by Poppendieck & Poppendieck (2006) which implies an ongoing development of the subject. Even though these are largely influenced by their manufacturing origin, Poppendieck & Poppendieck (2003, 2006) have clarified why these wastes are applicable in software development. These can be seen in Table 4.

The fact that lean software development is evolving is highlighted by Al-Baik & Miller (2014) who opposes Poppendieck & Poppendieck's (2003, 2006) definitions of waste in lean software development, since they are too heavily influenced by their manufacturing origin. They argue that no waste classifications have previously been developed purely based on an IT context. Thus, Al-Baik & Miller (2014) have developed a novel classification model covering both IT-operations and software development, showed in Table 4, based on a case study of an organizational unit of 250 employees. The model, with nine classes of waste, is partly different from the model by Poppendieck & Poppendieck (2003, 2006) and consequently also different from the wastes of manufacturing described by Toyota. However, since Al-Baik & Miller's (2014) waste model is based on one organization its generalizing potential may be negligible.

There is other research of waste in software development not influenced by Poppendieck & Poppendieck. Mandić, Oivo, Rodríguez, Kuvaja, Kaikkonen & Turhan (2010) identified Ohno's (1988) seven wastes in software development and added new sources of waste regarding decision making in software development: Avoiding decision-making, Limited access to information, Noise or information distortion and Uncertainty. These waste classifications can, however, be connected to Poppendieck & Poppendieck's (2003) work, where they argue that avoidance of decision making is valid (as in their LSD principle Defer commitment) since when uncertainty occur, delaying decisions as much as possible until they can be based on fact and not assumptions will generate better results. However, this will only be effective when able to act fast based on that decision (as in their lean principle Deliver fast). Moreover, Poppendieck & Poppendieck's (2006) waste category *Relearning* and *Handoffs* covers both of Mandić et al., (2010) wastes Limited access to information and Noise or information distortion. Thus, their research is both opposing and supporting

Poppendieck & Poppendieck's (2003, 2006) work while not bringing anything already discovered to the subject.

Korkola & Maurer (2014) conducted a study identifying waste of communication within globally distributed software development teams. They identified five wastes: lack of involvement, lack of shared understanding, outdated information, restricted access to information and scattered information. However, their study was aimed to specifically identify communication wastes in contrast to Poppendieck & Poppendieck (2003, 2006), Al-Baik & Miller (2014) and Mandić et al. (2010) findings that covers software development in general.

As the amount of research regarding lean software development is growing, the applicability of Poppendieck & Poppendieck's (2003, 2006) waste model has been being studied. Some research is directly applying their model in order to identify waste at case companies. For example, in Mujtba, Feldt, & Petersen's (2010) study waiting, extra processes and motion were identified when using value stream maps. Moreover, during a study by Ikonen, Kettunen, Oza, & Abrahamsson (2010) all of Poppendieck & Poppendieck's (2003) wastes was identified at some level which supports their model's applicability. However, it highlights that waste found in organizations cannot significantly explain if development is successful or not, but more so validates the waste model as successful in identifying improvement efforts.

The wastes found in various studies indicates that waste manifests itself differently depending on context. This is supported by Poppendieck & Poppendieck (2006) as they emphasize that the categories should not function as a classification tool, but rather work as a thinking tool facilitating understanding of the concept and thus reinforcing the habit of seeing waste. Moreover, Poppendieck & Cusumano (2012) means that much of the waste found in software development organizations is the result of "large batches of *Partially done work* created in sequential development processes, in the boundaries between different functions, and in the *Delays* and knowledge lost when work crosses these boundaries" (p. 28). They further state that causes of the waste can be identified and eliminated when organizations look at the entire value stream in an end to end perspective.

Table 4: Comparison of four waste classification models. Wastes organized in the same row have similarities, while waste on different rows have no clear relation. The comparison of the waste classification models is a suggestion by the authors.

Mandić et al. (2010)	Poppendeick & Poppendeick (2003)	Poppendeick & Poppendeick (2006)	Al-Baik & Miller (2014)
Inventory	Partially done work	Partially done work	-
Over-production	Extra features	Extra features	Gold plating
Extra processing	Extra processes	Relearning	-
Transportation	Task switching	Handoffs	-
Motion	Motion	Task switching	-
Waiting	Waiting	Delays	Waiting
Defects	Defects	Defects	Defects
Avoiding decision-making	-	-	Deferred verification and validation
Limited access to information	-	-	-
Noise or information distortion	-	-	Outdated information / obsolete working version
Uncertainty	-	-	-
-	-	-	Over-specifications
-	-	-	Lack of customer involvement and inappropriate assumptions
-	-	-	Double handling / duplicate processes
-	-	-	Centralized decision making

Partially done work

How organizations perceived inventory was forever changed when manufacturing sites adopted lean production and henceforth considered inventory wasteful. According to Poppendieck & Poppendieck (2006), the software development equivalent to inventory is *Partially done work* and should be considered an equal waste. Moreover, this can only be minimized if work items move to integrated, tested, documented and deployable code in a single rapid flow. This is however only applicable if work is divided into small batches or iterations. Poppendieck & Poppendieck (2003) means that *Partially done work* is harmful since it ties resources, like an investment, that have yet to yield results and satisfy customer

needs. Additionally, partially done software development carries a financial risk when uncertainty occurs whether the system actually will hit production and deliver the customer value intended. They further state:

> "…the big problem with partially done software is that you might have no idea whether or not it will eventually work. Sure, you have a stack of requirements and design documents. You may even have a pile of code, which may even be unit tested. But until the software is integrated into the rest of the environment, you don't really know what problems might be lurking, and until the software is actually in production, you don't really know if it will solve the business problem… Minimizing partially done software development is a risk-reduction as well as a waste-reduction strategy." (Poppendieck & Poppendieck, 2003, p. 5)

Poppendieck & Poppendieck (2006) gives examples of *Partially done work* as uncoded documentation (requirements), unsynchronized code, untested code, undocumented code and undeployed code which all should be kept to a minimum. Hence, *Partially done work* can be minimized by not releasing too much work into the development process and minimizing the number of tasks conducted simultaneously in the respective development phase. In Table 5, each of these examples is clarified.

Table 5 - Examples of Partially done work and the description of each. (Poppendieck & Poppendieck, 2003 & 2006)

Partially done work	Description
Uncoded documentation (requirements)	Written requirements that have yet not been coded will be more likely to change the longer the time before coding begins. Consequently, requirements should not be written too soon but instead written when needed by developers.
Unsynchronized code	Code must always be synchronized when developers commit their newly developed code into the code base. Synchronization should be done as frequent as possible since the longer code is separate, where many possible alterations possibly have occurred, the more difficult resynchronization will be.
Untested code	In order to detect and fix defects in code, a variety of tests is developed and executed. Testing should be done frequently when developing code, otherwise it will result in an exponential increase of Partially done work since defects in small volumes of code is easier to resolve. Performance indicators should measure progress only when the code is integrated, tested and accepted and enhance habits of regularly testing developed code.
Undocumented code	If documentation is needed, it should be done as the code is written and not after. In order to change old habits of doing the contrary, technical writers should be included in the development team instead of belonging to a separate unit.
Undeployed code	When the code is finished it should be deployed as soon as possible. It is often easier for users to absorb changes in small increments and customer value is achieved earlier rather than later.

Extra features

Poppendieck & Poppendieck (2006) state that *Extra features* can be considered the most harmful of the wastes in software development since it creates an exponential amount of waste in the system's lifespan. All code has to be tracked, compiled, integrated and tested and with each update of the system the scope of these activities increases for every bit of extra code (Poppendieck & Poppendieck, 2003). Hence, these *Extra features* need to tend to continuously and becomes useless waste draining resources from a number of more important activities. Like any code, *Extra features* are potentially becoming a weak link in a system with safety or stability issues as a potential outcome. Moreover, Poppendieck & Poppendieck (2006) mentions that unnecessary code creates wasteful complexity and increases the difficulty to execute changes safely. Consequently, developers top priority should be on keeping the code base simple and clean and resist the temptation of developing features not requested at the moment.

The most effective way of decreasing complexity is by limiting features to enter the code base, to begin with. Every feature that is developed should be able to provide more economic value compared to its lifecycle cost. This way of always being critical to features takes courage, but it pays for itself many times over. (Poppendieck & Poppendieck, 2006)

Relearning

According to Poppendieck & Poppendieck (2006), the concept of *Relearning* can be described as rediscovering something once known but forgotten and can be seen as the manufacturing equivalent of rework. However, waste in knowledge can also be identified when people bring knowledge to the workplace but the organization fails to engage that knowledge in the development process. *Relearning* can be considered the inverse of captured knowledge and described by Poppendieck & Poppendieck (2006) as:

> "In software development, the tests and the code are often just the right combination of rigor and conciseness to document the knowledge embedded in the software. But experiments tried and options investigated on the way it making decisions about the product under development are easily forgotten, because they never make it into the software. And just writing it down does not necessarily mean that the knowledge is saved, since information is just as easily lost in a sea of excess documentation as it is lost through lack of documentation." (Poppendieck & Poppendieck, 2006, p. 156).

It is a common thought that writing down information during an iteration will contribute to organizational learning, which is wrong since most of the documentation will be untouched and only fill disk space (Poppendieck & Poppendieck, 2006).

Handoffs

Tacit knowledge is difficult to capture through documentation where every following Handoff further decreases the amount of tacit knowledge. Poppendieck & Poppendieck (2006) mentions that *Handoffs* should be kept at a minimum and development should be done by teams covering all necessary functionality. They further mention that "high-bandwidth" communication is recommended in contrary to documents and an increased amount of knowledge can be kept between *Handoffs* if work is released partial for consideration and feedback, as soon as possible and as often as practical. Consequently, when the different software development functions are loosely integrated, it is expected that *Partially done work* is accumulating between the functions resulting in problematic *Handoffs*. (Poppendieck & Poppendieck, 2006)

Task switching

Software development takes concentration whereby switching from one task to another requires time to reset. According to Poppendieck & Poppendieck (2006) when executing work that requires concentration and having to many tasks the time to reset between these tasks can take more time than the actual time spent on working with the tasks. Thus, belonging to multiple projects is ineffective since it will increase the number of interruptions that occurs (Poppendieck & Poppendieck, 2003). Moreover, work will move faster through a process not filled to its capacity which is the opposite of running multiple projects. Thus, a development team should not start several projects at a time and the organization should resist the temptation of releasing too much work into the development process.

Delays

According to Poppendieck & Poppendieck (2006), developers make critical decisions regularly where gathering the necessary, but not accessible, information often creates *Delays*. *Delays* are common in most software developments whereby it is regularly perceived as something natural, at worst (Poppendieck & Poppendieck, 2003). Waiting is an output of *Delays* and should not be perceived as something natural but rather as a serious waste. Poppendieck & Poppendieck (2003) means that *Delays* can potentially occur at a number of activities; staffing, requirements documentation, reviews, testing, deployment etc. which all keeps work from moving downstream and realizing the value to the customer quickly. Furthermore, an organizations ability to fulfill critical customer request rapidly are in direct correlation to *Delays* in the organization's development processes. A fundamental lean principle is to take decisions as late as possible in order to make the most informed decisions. However, this is not applicable if *Delays* are stopping the rapid implementation of these decisions. Thus, *Delays* should not be tolerated and organizations should constantly try to analyze why *Delays* happens and if possible eliminate root causes.

Defects

According to Poppendieck & Poppendieck (2003) the amount of waste generated by *Defects* are related to the impact of the defect, but even more associated with the time *Defects* goes unnoticed. Critical *Defects* that are resolved quickly are consequently less wasteful than minor *Defects* undiscovered until it reaches the customer. Waste generated by *Defects* can be minimized through immediate testing and frequent integrations in order to establish a situation where *Defects* found in verification is not routine but rare (Poppendieck & Poppendieck, 2006). Furthermore, whenever a defect is found, a test should be created so the process becomes mistake proofed in time. However, the real benefit of moving testing to the beginning of development is because it constitutes how developers expect the code, and the product, to work.

Poppendieck & Poppendieck (2006) recommends that developing teams should support their own code since it provides motivation to deliver defect-free code and cannot possible push the problem to a maintenance team. This can seem counterintuitive but in time defect free code results in less *Task switching* for the development team instead of a constant *Task switching* for a maintenance team.

3.3 Waste elimination strategies

Wang, Conboy, & Cawley (2012) examined 30 experience reports published in past agile software conferences in which lean approaches in agile software development were reported. In many of these experience reports, waste elimination was a lean element applied. However, how waste was identified and which waste elimination strategies used is not explained. Al-

Baik & Miller (2014) picks up on this topic and means that research considering how to identify and eliminate waste are lacking whereby the authors provides a detailed description of a lean software development initiative in an industry case, including successful waste identification and elimination strategies. In their study, Al-Baik & Miller (2014) identified 42 different wastes (in nine different categories summarized in Table 4) with an equal number of elimination strategies. Al-Baik & Miller (2014) states that "In order to have a successful Lean journey, enough time must first be allocated to analyze and understand the nature of the organization, its business processes, and the environment in which the organization operates" (p. 2021). The context specific elimination strategies successful in Al-Baik & Miller's (2014) study supports the fact that waste should be managed individually and no one solution fits all is advocated, but instead a habit of continuously identifying and eliminating should be incorporated in the organization. This is supported further by Poppendieck & Poppendieck (2006) who mean that classification in itself is insignificant but the lean thinking mindset is of importance. Moreover, Al-Baik & Miller (2014) agrees and argues that identification and elimination may be done iteratively and a lean mindset is the shortest path for a successful lean journey. Poppendieck & Poppendieck (2006) states that only principles are universal while practices have to be developed with respect to the internal organizational environment in order to be successfully implemented.

The findings of Al-Baik & Miller (2014) highlights the importance of senior management and employee support when eliminating waste. Senior management should understand the value of investing in different lean initiatives and employees needs to understand the benefits lean can stimulate in their own work environment. Consequently, Al-Baik & Miller (2014) means that quick and significant improvements must be the result of early initiatives in order to convince the whole organization of the importance of these lean activities.

Petersen & Wohlin (2010) highlights the two lean software development principles waste elimination together with a holistic view of the process as facilitating when improving software development processes. However, the authors also recognize the large shift in thinking about the organization's software development processes required to adopt lean. Hence, the change to lean has to be done in a continuous and incremental way.

3.4 Continuous Improvements

In order to make an improvement, organizations must do changes in their current working methodology. However, in order to continuously improve, the organization is required to constantly examine itself (Klefsjö, Eliasson, Kennerfalk, Lundbäck, & Sandström, 2010). Swaminathan & Jain (2012) emphasizes that the concept of continuous improvement not only is applicable to software development but also contributes to significant benefits. During Swaminathan & Jain's (2012) study, they could reveal that it is possible to conduct a software development project with a smooth flow, which additionally facilitates continuous improvement. Miller & Al-Baik (2016) conducted a study regarding how to sustain the benefits obtained from implementing the lean principle Eliminating waste. In their study, continuous learning and improvements resulted in significant benefits to the organization. Four different approaches to realize continuous learning and improvements were tested at the organization: Reflective practices, 5 Whys, Policies and Standards and Double-loop learning.

Reflective practice regards the reflection of previous experiences and is divided into two branches: reflection in action and reflection on action (Miller & Al-Baik, 2016). Reflection in action is instinctively made improvements that are based on previous experiences. Moreover, these improvements are done instant as the action is happening. Reflection on action concern

the investigation of past experience to identify potential pitfalls and thereby ensure future improvements and success.

5 Whys is a factually based approach, utilized in order to identify the root cause of a problem (Murugaiah, Benjamin, Marathamuthu, & Muthaiyah, 2010). By asking the question why, five times concerning a single problem, identification of a root cause is possible that may prevent a problem from reoccurring and thereby improve the process or task conducted (Miller & Al-Baik, 2016). Additional to the root cause identification 5 Whys also contributed to reflection in action, during the implementation of the method (Miller & Al-Baik, 2016).

Policies and Standards are constructed in order to minimize faults made by the employees (Kondo, 2000). Kondo (2000) argues that when anomalies occur in a process, these should be fixed and the standard should be updated, which indicates that an improvement has been realized. However, standards should not be enforced upon workers without an associated declaration regarding the goal of the standard (Kondo, 2000). Otherwise, the sense of responsibility among the workers diminishes, making them only work towards the standard and not the aim of the standard. Miller & Al-Baik (2016) thereby recommend the self-determined standard. This standard contributes to continuous improvement since the implementer itself can upgrade the standard if new improvements are found, thus creating a new standard (Miller & Al-Baik, 2016). Moreover, the organizational memory can be enhanced by utilizing self-determined standard, if it is documented properly.

Double loop learning regards questioning the existence of tasks and problems (Miller & Al-Baik, 2016) instead of solely questioning how to perform a task or find a solution to the problem. Consequently, organizations can avoid spending resources on activities that should not have been performed in the first place. Argyris (1994) means that Double loop learning facilitates the ability to change the current values that lead to a contra productive behavior. This questioning approach is important to possess when conducting a PDSA cycle, which Bergman & Klefsjö (2013) considers as the symbol of continuous improvements.

3.4.1 PDSA-Cycle

The PDSA-cycle is a continuous improvement tool, created by William Edwards Deming in 1986, which was an extension of the Shewchart cycle created in 1939 (Deming, 1986). The cycle is identified as a flow diagram for learning and improvement of a process or product (Deming, 1993). The PDSA cycle is presented in Figure 3.

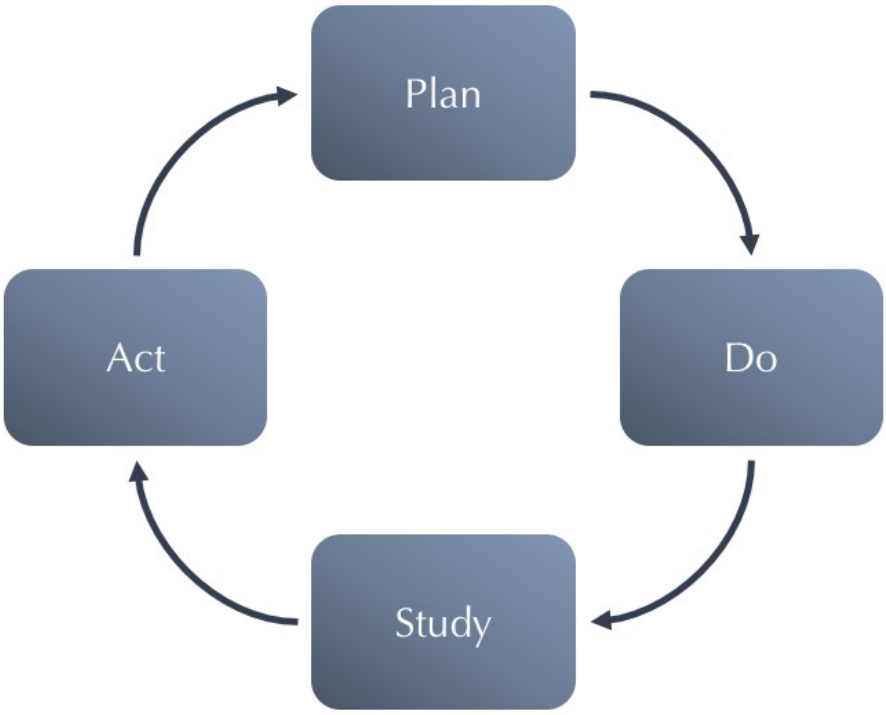

Figure 3: The PDSA- Cycle consists of four different steps, these steps are Plan, Do, Study and Act (Deming, 1993). Adapted form: (Deming, 1993, p 135)

STEP 1, PLAN: An idea for an improvement of a product or a process is discovered, leading to a plan for test or experiments (Deming, 1993). The first step is perhaps the most important one and is seen as the basis of the entire cycle, where Deming (1993) further state that a hasty and non-thought beginning ends up as costly and ineffective.

STEP 2, DO: In this phase, the practitioners execute the test or experiments chosen from step 1, preferably on a small scale (Deming, 1993).

STEP 3, STUDY: Study the achieved results from step 2. Examine if the result corresponds to the earlier expectations, and if not, analyze what went wrong (Deming, 1993). Occasionally, the error lies in the planning phase and therefore restarting the cycle may be a good option.

STEP 4, ACT: In this phase, the practitioners should either implement the change within the organization, skip it or restart the cycle once more, but with different conditions than the previous run (Deming, 1993).

3.4.2 Measures relevance in continuous improvements

In order to achieve continuous improvement in a software development project, measures and metrics is a significant element (Swaminathan & Jain, 2012). Bergman & Klefsjö (2013) state that decisions should be made based on facts, involving structuring and analysis of certain information. Further, Bergman & Klefsjö (2013) argue that systematic collection of information regarding customer needs and desires is needed to obtain customer focus. Russell (2003) continues on this topic and argues that inspections are needed in order to collect the desired data, upon which long-term strategic decisions can be made, as well as reinsuring the

projects satisfactory completion. Furthermore, Russell (2003) state that implementation of continuous improvements leads to an enhanced long-term health of an organization. Regarding measures and metrics, Staron (2012) recognized that the ability to control, monitor and predict the result of software engineering processes is of great importance in software development organizations. Staron (2012) further argue, that effective use of measurements does not require a large number of metrics, however the metrics need to be clear, reliable and automatically collectable. Only a few key indicators supported by measures regarding appropriate statistics and trends should be known to the whole company (Staron 2012). Moreover, Staron (2012) mean that the usage of excessive metrics has a potential of becoming wasteful in organizations, since they do not provide value for the decision making processes.

3.5 Lean measures

Petersen (2012) highlights the importance of indicators and measures in order to facilitate implementation of lean principles through continuous improvements. However, measures for lean software development should not be considered in isolation, but instead visualized and analyzed combined in order to achieve a holistic perception of the situation and thus minimize the risk of sub-optimization. Petersen & Wholin (2010) means that the combined analysis together with system thinking enables the organization to find root causes where the impact of problems or improvements on the overall process should be taken into consideration. Common process characteristics to measure in lean is flow and lead-times, however, Petersen & Wholin (2010) argues that these should be considered in combination with quality. This enables decisions to be made that increases the performance of the process without resulting in a lesser quality of the products.

Petersen & Wholin (2010) argues that *Partially done work* should be in focus when improving the software development process since high inventory levels indicate waste and an absence of a lean process. Moreover, inventory hides *Defects* (Middelton, 2001), increases the risk of changes making work obsolete, creates others wastes like waiting (Petersen, Wohlin, & Baca, 2009), disturbs flow which causes overload situations (Petersen & Wholin, 2010) and causes stress in the organization (Morgan, 1998). Figure 4 illustrates the *Partially done work* generated in software development represented by stacked boxes. In the center of the figure, the generic software development process with respective process phase constitutes Normal work, generating *Partially done work* between each phase. Extra work is changing requests from the customer and faults identified either by testing or by the customer when reviewed and can also be considered as unplanned work. The top stack of boxes is the quality parameter as Fault slip through, also known as escaped defects. Figure 4 is a modified version from Petersen & Wohlin (2010) with influences from Poppendieck & Poppendieck (2003, 2006) who argues that undeployed code should be considered as *Partially done work*.

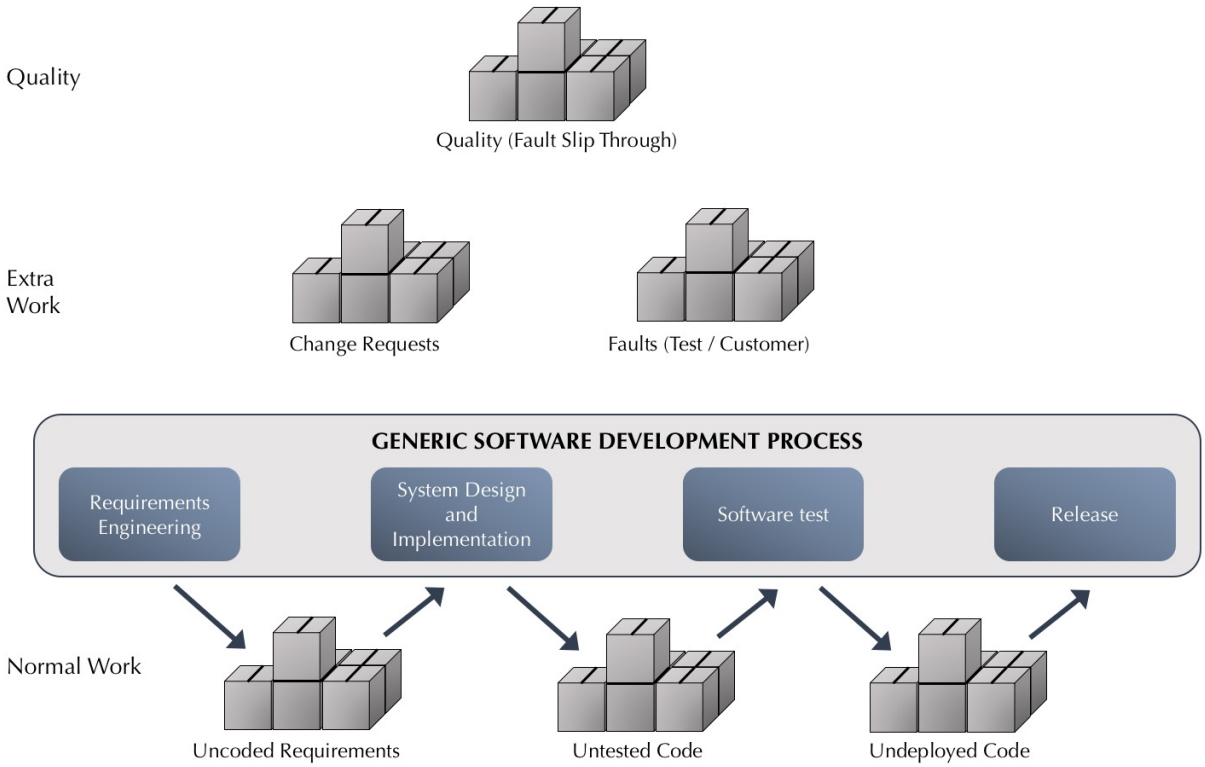

Figure 4: Inventory for Partially done work in software development
Modified from Petersen & Wholin (2010), influenced by Poppendieck & Poppendieck (2003, 2006)

In order to indicate the process flow and lead time and incrementally improve the process, measures tested in case of studies at Ericsson by Petersen & Wholin (2010) and Petersen (2012) showed potential for identification of waste. In these case studies, requirements by inventory levels in consideration to capacity (Statistical process control), flow (cumulative flow diagram) and lead time (box-plots) was some proposed measures that showed potential in practice. Moreover, these measures were accepted and supported by the case company because of the minimal prerequisites for implementation. Petersen (2012) found that the case company was able to make a comprehensive analysis and identify inefficiency while only having to keep track of a few measures. Other organizations that would like to utilize these measures needs to collect three dimensions of data: registration of work items with time stamps, state-changes of work items in the process, and classification of the work item if necessary to distinguish between them.

3.5.1 Inventory levels (workload)

The level of *Partially done work* in a process should be considered in relation to its capacity, in order to attain smooth flow in the process (Petersen & Wohlin, 2010). Moreover, avoiding overload situations also shows respect to the employees, as in the lean principle respect people, and ensures that the motivation is high (Petersen, 2012).

Inventory levels can be monitored with statistic control charts, namely data points plotted with regards to mean and control limits of +/- 3 standard deviations over and under the mean, according to Petersen & Wholin (2010). Moreover, empirical evidence from Petersen & Wholin's (2010) study showed situations where inventory levels were over the upper control limit, developers felt overloaded and thus no refactoring occurred. The opposite was observed when data points were inside the control limits, a situation where most requirements passed

testing and were ready for release and thus the developers could spend time on activities such as refactoring. Petersen & Wholin (2010) means that this is a good basis for further discussion of capacity, which is further supported by findings in (Petersen, 2012). Moreover, the practitioners in Petersen & Wholin's (2010) study agreed that work-load should be below full capacity since it not only enables more steady flow but also increases flexibility to fix problems and unplanned work like bugs or customization requests. Complementing the control chart of inventory levels, data points can also be visualized by the moving range and thus indicate batch-behavior which also constitutes a risk for overload situations in specific development phases, according to Petersen & Wholin (2010).

The choices of inventory that constitute data points are multiple: Normal work, extra work, defects, maintenance etcetera, and will influence what information that can be extracted from the charts. For example measuring the level of *Defects* or maintenance requests will not only enable indication of workload, but also quality (Petersen & Wholin, 2010), while measuring all work items combined can give a complete picture of the workload, but does not indicate if the most prioritized type of work items are getting the attention needed (Petersen, 2012). Since the priorities and problems will differentiate between organizations, what inventory that constitute data points will require individual customization based on the organization's needs. However, multiple control charts enable more comprehensive analysis when needed (Petersen, 2012).

In order to appropriately indicate the workload a specific level of *Partially done work* in the process represents, work items should be classified based on complexity since a complex problem would likely cause more workload compared to an easy problem (Petersen, 2012). The same argument could be done regarding the size of work items, for example, based on how many lines of code, since it influences the lead time of the work item, as shown by Petersen (2010). Practitioners in the case study conducted by Petersen & Wholin (2010) also recognized the high variance in software development work items and supported the usability of classifying work items depending on size. When using the above-mentioned classifications with the organization's own thresholds, the workload can thereafter be adequate estimated by multiplying more time-consuming work items by a factor. Moreover, Petersen & Wholin (2010) highlights the importance of dealing with the possibility of local optimization of the measures. For example, in order to reduce *Partially done work* in one development phase, practitioners could improve their measurement by cutting corners and quickly hand over work items to the next development phase. The solution is twofold according to Petersen & Wholin (2010), a measure of quality related inventories and measure process flow in order to indicate batch behavior.

3.5.2 Flow

Petersen & Wholin (2010) highlights the importance of a continuous flow of requirements in software development and suggest cumulative flow diagram as a suitable indicator. This enables more detailed analysis of inventory where *Handoffs* and *Partially done work* in specific development phases are more clearly visualized. Accordingly, cumulative flow diagrams can display if development is conducted in small and continuous increments (Petersen & Wohlin, 2010). This will, in turn, enable identification of bottlenecks and other waste according to Petersen & Wohlin (2011) and is, therefore, a foundation for continuous improvement. Moreover, cumulative flow diagrams have found support from practitioners in case studies conducted at Ericsson by Petersen & Wholin (2010), Petersen & Wholin (2011) and Petersen (2012). Practitioners acclaimed the model since it is easy to use and useful in influencing management decisions. Moreover, Petersen & Wholin (2011) means that the measures were integrated quickly in the practitioners' work practices and improved their:

requirements prioritization, staff allocation, problem and improvement identification; and transparency of current status. The increased transparency is especially beneficial in the development of complex products with many tasks in parallel, according to Petersen, Wohlin, & Baca (2009).

The cumulative flow diagram consists of data point describing *Partially done work* in each development phase plotted over a specific time window (Petersen & Wohlin, 2011). A sketch of this is shown in Figure 5 with terms from Petersen & Wholin (2011).

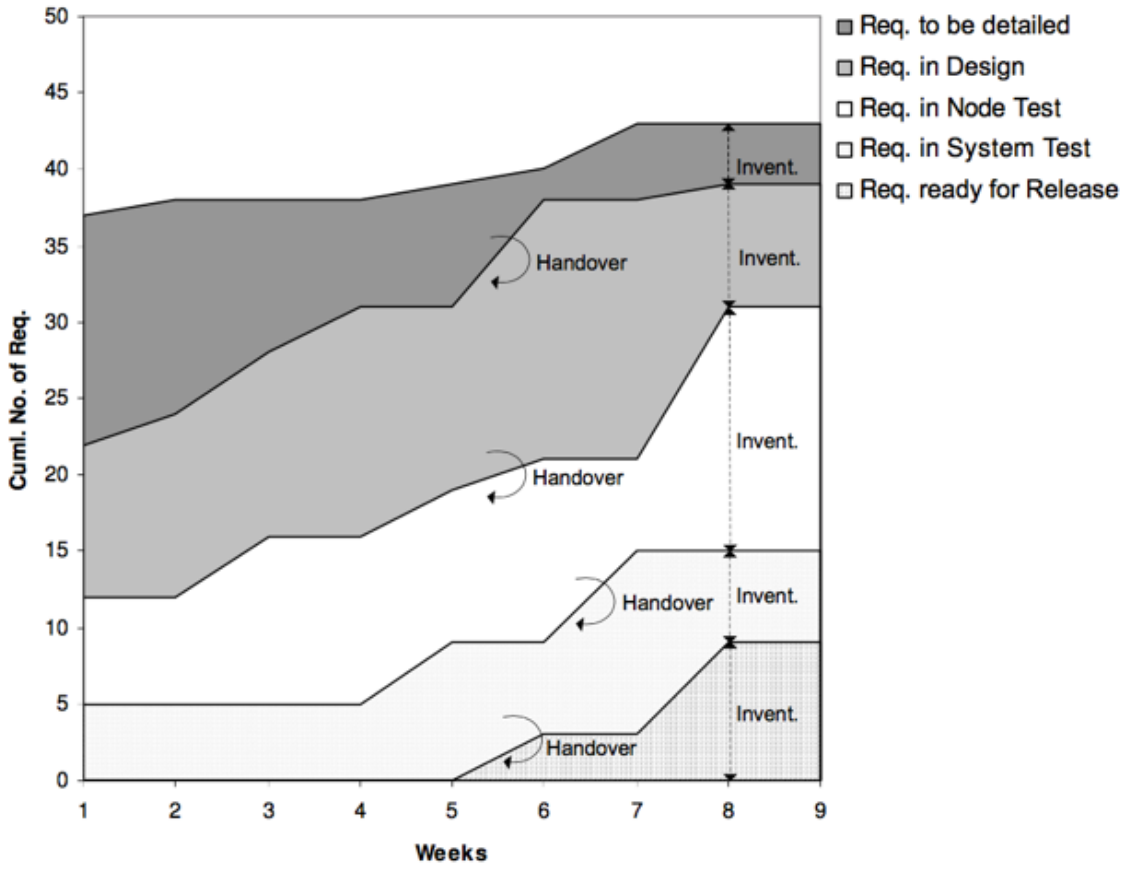

Figure 5: Illustration of a Cumulative flow diagram
Source: Petersen & Wholin (2011)

The y-axis represents the cumulative number of work-items that have completed different phases in the development while the x-axis represents the timeline. The inflow of new work items are added to the top, that represents the first phase of development, and when work-items are progressing in the development process, they are consequently flowing downwards in the diagram. Moreover, the top line represents the total amount of work-items currently in the process while the line segments represent a number of work-items in each respective development phase. Further, the slope of the lines indicates if handovers are done continuously or in a batch pattern since large handovers are more quickly changing the level of requirements from one phase to another while the overall level of work-items stays the same (Petersen & Wohlin, 2011).

A bottleneck in the software development process is defined by Petersen & Wholin (2011) as a phase where work-items enter at a higher rate than handed over to the next phase. This causes overload situations and should be avoided. Bottlenecks can be found in the cumulative flow diagram by visually analyzing the slopes and number of *Partially done work* over time,

in the different development phases. However, a visual analysis will not always result in the right conclusions, why Petersen & Wholin (2011) propose linear regression model to measure the rate of work-items flow in each phase.

3.5.3 Lead time

Petersen (2010) states that short lead-times are essential in fast-paced markets. However, short lead-time comes from a fundamental understanding of how lead-times are affected by decisions and thereby taking the right actions. Carmel (1995) confirms this approach by stating that the awareness of lead-time is important in order to choose the right actions. Moreover, the study confirmed that team factors such as cross-functional, motivation and team size are critical in order to minimize lead time.

Measuring the lead-time supports the lean software principle Deliver fast, according to Petersen (2012). However, based on the nature of software development, lead-time is generally affected by large variances in this context. Petersen (2012) continues by mentioning that the lead-time through different phases also should be subjected to variances. Consequently, lead-time should be analyzed with regard to the natural distribution of lead-time of the environment. In order to favorably visualize the measures of lead-time, with regard to the above-mentioned variances, box-plots is suggested by Petersen (2012). Lead-time measures can further give a complete perspective of the situation by comparing lead-times between high priority and low priority tasks in order to indicate the process effectiveness. Furthermore, distinguishing between value adding and waiting time can indicate improvement efforts regarding the lead time according to Petersen (2012). A tool that accomplishes this distinction between waiting and value adding time is Value Stream Mapping (VSM), a method highly recommended by for example Poppendieck & Poppendieck (2006) and Petersen (2012), appreciated for its ability to identify improvement efforts.

Value Stream Mapping

Mujtaba, Feldt & Petersen (2010) describes a value stream as “all the actions (both value added and non-value added) currently required to bring a product through the main process steps to the customer also known as the end-to-end flow of process” (p. 139). Value Stream Mapping is one practice that can be used in order to eliminate waste with a complete value stream perspective, thus complying with the lean principle optimize the whole described by Poppendieck & Poppendieck (2006). Moreover, the strength of VSM is its ability to facilitate organizations with an understanding of any workflow with an end-to-end perspective. This is further supported by Petersen (2010) who means that by activating practitioners from multiple sources, e.g. teams and units, it requires practitioners to consider the entire value stream. Consequently, it decreases the risk of improvements considered as sub-optimizations. The course of action when applying VSM is described in detail by Poppendieck & Poppendieck (2006) and McManus & Millard (2004).

3.6 Summary of waste elimination approaches in literature

The literature review provided multiple approaches to waste elimination. These approaches were summarized in Table 6 where each approach is structured in either of the four aspects: awareness, indication, analysis and elimination.

Table 6: Summary of waste elimination approaches in literature.

Aspect	Approach
Awareness	Understanding the concept waste (Poppendieck & Poppendieck, 2003; Al-Baik & Miller, 2014)
	Classification model creates a mindset, reinforcing the habit of seeing waste (Poppendieck & Poppendieck, 2003; Al-Baik & Miller, 2014)
	Senior management and employee support (Al-Baik & Miller, 2014)
Indication	Clear, reliable and automatically collectable measures (Staron, 2012)
	Provide foundation for analysis (Petersen & Wholin, 2010)
	Visualized (Petersen, 2012)
	Partially done work in consideration to capacity (SPC) (Petersen & Wholin, 2010; Petersen, 2012)
	Flow (cumulative flow diagram) (Petersen & Wholin, 2010, 2011; Petersen, 2012)
	Lead time (box-plots) (Petersen & Wholin, 2010; Petersen, 2012)
	VSM (Poppendieck & Poppendieck, 2006; Petersen, 2010, 2012)
Analysis	5 Whys (root cause analysis) (Miller & Al-Baik, 2016)
	Indicators analyzed combined (Petersen, 2012)
	Indicators considered in combination with quality (Petersen & Wholin, 2010)
	End to end perspective of value stream (Poppendieck & Cusumano, 2012; Petersen & Wohlin, 2010)
Elimination	Practices adapted to context (Poppendieck & Poppendieck, 2003; Al-Baik & Miller, 2014)
	Polices and standards, self-determinded, continuously improved (Miller & Al-Baik, 2016; Kondo, 2000)
	Identification and elimination iteratively (Al-Baik & Miller, 2014; Petersen & Wohlin, 2010)
	Continuous improvements (Swaminathan & Jain, 2012; Miller & Al-Baik, 2016)
	Reflective practices (Miller & Al-Baik, 2016)
	Double-loop learning (Miller & Al-Baik, 2016)

4 Case organization

This chapter will present a brief review of Infor and Infor M3 and information about the organizational structure of Integration & BI. The information regarding the Integration & BI was gathered through observations and interviews conducted during the case study.

4.1 Infor

Infor is an enterprise software provider consisting of more than 40 acquisitions brought together by the private equity firms Golden Gate Capital and Summit Partners in 2002 (Lev-Ram, 2015). With over 15,000 employees in 41 countries, delivering software to over 90,000 customer organizations (Infor corporate overview, 2016), Infor is the third largest enterprise software provider worldwide (Lev-Ram, 2015). Infor focuses on a variety of industries and has contracts with some of the largest actors in these respective industries, presented in Figure 6.

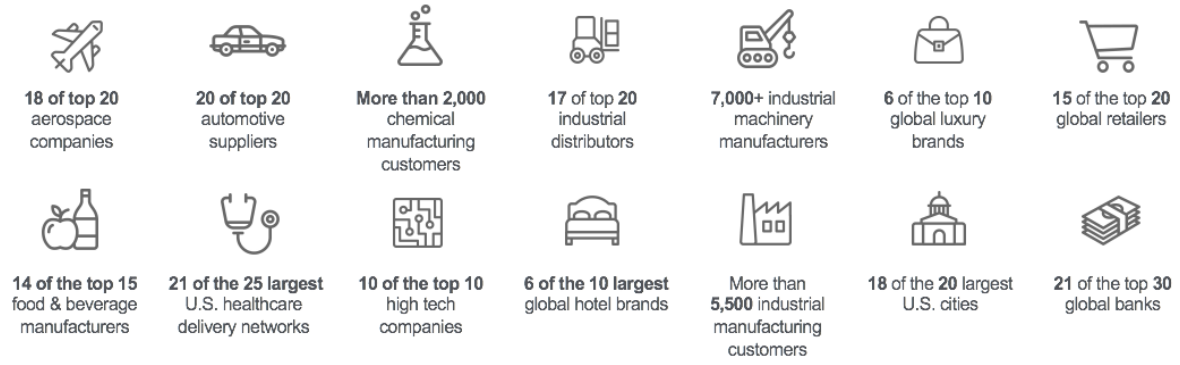

Figure 6: An illustration of the different industries where Infor´s customers are active
Source: Infor corporate overview, 2016

Infor's over 40 acquisitions result in a diverse software solutions portfolio where systems can be connected and thereby create complete business management platforms, called suites, specialized for different industries. For the customers, this implies that they do not need to invest in several systems from different providers and tailor them together in order to reach the functionality requirements, but instead have one provider that delivers a complete package tailored to the specific industry needs. Infor describes their suites as:

> "Our software is purpose-built for specific industries, providing complete suites that are designed to support progress – for individuals, businesses, and across networks. We believe in the beauty of work, the importance of relationships, and the power of ideas to drive significant positive change." (Infor corporate overview, 2016, p.2).

Many software providers face the transition from on premise to cloud based solutions and Infor is no exception. Infor's strategy of collecting multiple systems under the same organizational umbrella facilitates the options for customers to attain a complete set of enterprise software from one single provider delivered as SaaS. Infor is already delivering multiple of their business suites as SaaS solutions to customer and internally uses the motto "*Cloud First*", which means that work regarding getting their services to the cloud is prioritized. Infor is thereby in the middle of this transition, where their main focus is to translate and deliver their arsenal of solutions to the cloud, delivered as state-of-the-art SaaS solutions, including Product Lifecycle Management (PLM), Human Capital Management

(HCM), Enterprise Resource Planning (ERP), Supply Chain Management (SCM) etcetera (Infor Corporate Fact Sheet, 2017).

One of the ERP systems owned by Infor is Infor M3. The Swedish company Intentia has developed M3 since the 80's until 2005 when Lawson acquired Intentia. Infor acquired Lawson in 2011 and thereby the M3 system. Today, M3 covers all main business processes for manufacturing, distribution and maintenance companies like an order to invoice processes, resource planning, purchase, and maintenance while also having an integrated financial system with accounts receivable, accounts payable and general ledger support. Infor M3 is currently serving about 1,200 customer enterprises and has over 300,000 users worldwide.

M3 has spent the last two years focusing on developing a multi-tenant solution of their software, while also maintaining on-premise and single tenant cloud versions of the software. As an increasing amount of customers are interested in a SaaS solution of M3, the multi-tenant solution is vital in order to reach cost efficiency.

4.2 Organization at Infor M3 Integration

The subject of the case study in this research is Infor M3's Integration & BI unit and the associated teams, which is presented in Figure 7 below with green framing.

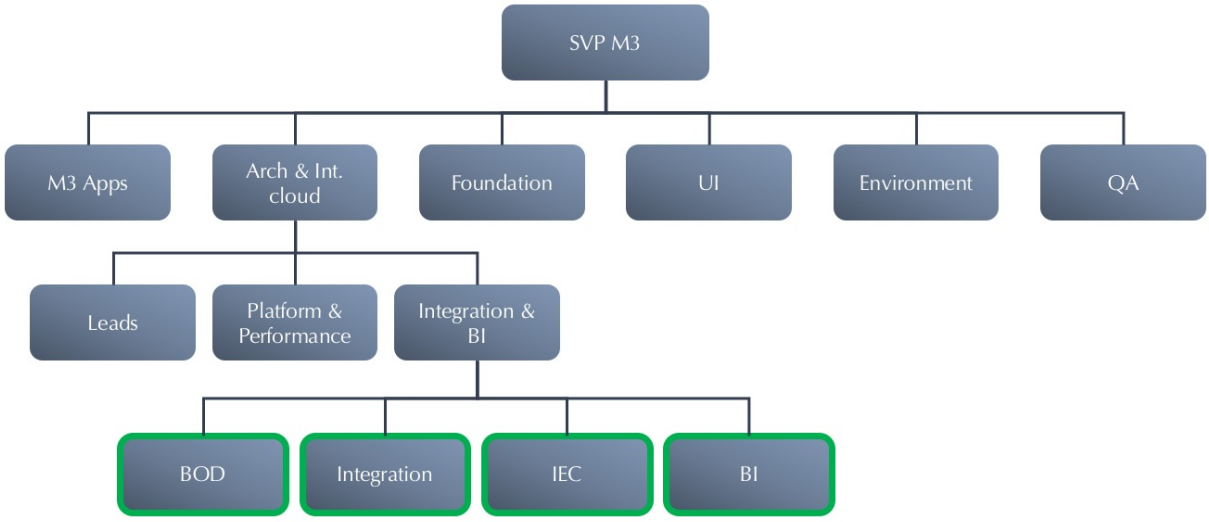

Figure 7: Organizational position of Integration & BI at Infor M3. The associated teams of this case study is highlighted by a green framing.

Infor's strategy to connect their acquired systems into suites and deliver these to customer enterprises as SaaS makes integration a fundamental keystone in order to fulfill Infor's overall vision. Any of these integrated suites have a foundation of one system, like the ERP-system M3. However, include functionality from a number of other Infor products that needs to be connected and work together as one product. M3 Integration & BI is responsible for developing and maintaining these integrations between Infor M3 and other Infor software and ensure that these functions in a satisfactory way. Since many of Infor's products have overlapping functionality, while still complementing each other into better complete solutions, it is important to sort out what functions should be handled by which system in the suites. This makes the responsibility of Integration & BI complex since many technologies and parties needs to be coordinated. In the case of Integration & BI unit, the choice of which integration projects to undertake is important since the demand of integrations to M3 is

higher than the unit's ability to run integration projects. Consequently, it would be beneficial if their throughput could be improved and thus leave room for more integrations. Moreover, all teams associated with Integration & BI unit are not developing integrations, but rather the actual software that makes the integrations possible. This presents a challenge for the unit where the team's agendas differ: some teams wanting to improve the software in order to make future integrations easier, but are interrupted by integration projects; while other run integration projects that are delayed when the underlying software is not adequate.

The teams in the Integration & BI unit have diverse responsibilities ranging from developing and maintaining actual software products that enable connection between M3 and other Infor systems, to managing integration projects. The unit categorizes the teams' responsibilities in technology and content, where technology is software and tools that enable communication between systems and content as the data transferred between the products and their design. Consequently, many of the teams are dependent not only on other integrated systems but also to other teams within the unit. Some of these teams focus on creating a software product while others are set on creating an actual integration. These will, in turn, be depending on the underlying product and their capability to provide appropriate tooling to enable these integrations.

Communication in integrations between Infor M3 and other Infor products is performed by business object documents (BODs) that enable asynchronous exchange of information from one system to another by an XML master pattern. The design of the BODs themselves are developed by the BOD team and retrieves data from the Infor M3 system by application programming interfaces (APIs). This is in turn managed by Infor M3 BE (Business engine), the unit developing the Infor M3 system and thus outside the Integration & BI unit's management. BODs are the core of integrations and best described as a mode of transportation, but needs to be mapped (connected logically) differently depending on the integrating system, which is performed in the administration and configuration interface IEC. The IEC team develops and maintains the product that is used by both customers in order to customize their integrations, and by the Integration team who develop new integrations by mapping the BODs in the IEC interface. However, the responsibilities of the Integration team are more diverse than only the actual construction of integrations by mapping BODs. Much of the work performed in the Integration team can be categorized as project management, since integration projects require synchronization of a number of resources internally in the unit, in M3, but also externally. Integrations must be co-developed with the other software M3 integrates with and requires solid by-in and detailed specifications of the scope of the project in order to align both organizations. Moreover, new integrations generally require new BODs or changes in existing BODs. Thus the requirements (tasks) of the BOD team is mostly driven by the projects the integration team is pursuing. In turn, new integrations, with respective new BODs or BOD changes, may also need API changes managed by BE. In summary, it can be concluded that the above-mentioned teams of the Integration & BI unit, namely BOD, IEC, and Integration, are connected and that integration projects will trigger requirements in many teams, but also in external units.

The BI team is somewhat isolated from the other teams of the Integration & BI unit while still organized in the same business unit. This is because BI solutions can be recognized as a function extracting data from multiple systems in a suite and thus operates in the boundaries between systems. The BI team's product functions to support collection, analysis, and presentation of business information in order to provide an overview of enterprises business operations. Most of the development the team perform are solutions to organize data sets from the customers' enterprise systems and present these in so-called widgets, customizable to suit the customer needs.

Infor M3's Integration & BI unit has employees located globally with the majority stationed in Sweden and the Philippines and a few located in USA and Germany. Moreover, the employees in Sweden and the Philippines are scattered nationwide which makes communications via electronic mediums like Skype and email a vital part of the units work environment. In general, employees in Sweden have senior experience with the system and holds a coordinator role in each respective team while the bulk of the development and maintaining is performed in the Philippines, with a more production-oriented approach. The location of each team's employees is presented in Figure 8.

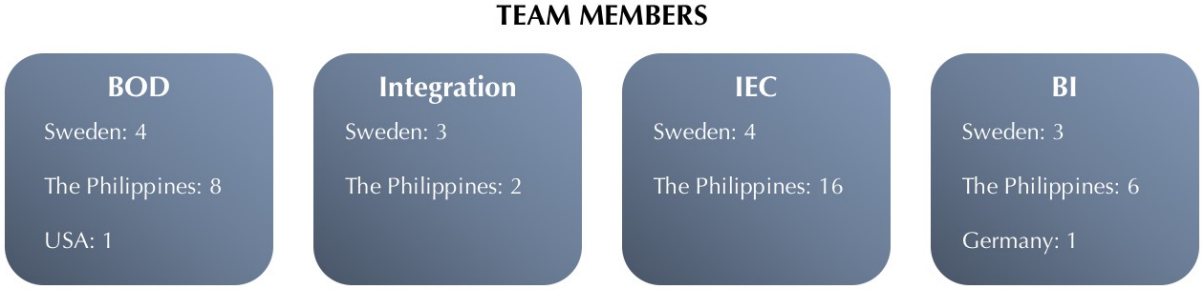

Figure 8: The location of Integration & BI's team members

The transition from delivering M3 on premise by license to SaaS have introduced new challenges to M3 regarding responsiveness and flexibility but also regarding security and scalability. Delivering a SaaS solution implies that Infor is responsible for the software's maintenance, upgrades and operation in contrast to an on-premise system where these activities are taken care of by the customers own IT-operations department. Subsequently, new demands regarding the simplicity and automation of upgrading and maintenance are introduced since Infor becomes responsible for executing these activities for a large customer base. This increases the need of effective work prioritization and efficient development processes.

The vision for M3 is to be able to deliver flawless products to the customers as often as possible, desirable in small increments every other second. The transition to multi-tenant SaaS solutions has triggered a change in Integration & BI's software development methodology from a more waterfall oriented development approach to an agile approach. However, the successfulness of the teams' transition to an agile methodology is not consistent and correlated to both habits and technical complexity of the products. Moreover, the inconsistency in the different team's prerequisites for working incrementally disabled the use of a centrally decided working methodology (e.g. Extreme Programming, Scrum, Kanban, Lean and DevOps etc.). Instead, they are themselves required to figure out what is best suited in their context. However, in order to achieve a more flexible and responsive organization, the teams are currently required to have a sprint approach, ranging from one week to four weeks, and are evaluated based on how well these sprints is executed according to planning. The evaluation is done every four weeks, in which each team is measured by a number of Key performance indexes (KPI):

- Commit to done (How much of planned work completed by end of sprint)
- Open defects (discovered by the team, equivalent to rework in manufacturing)
- Escaped defects - QA (defects discovered by a central M3 QA organization, before release)
- Escaped defects - customer (complaint, may include risk of monetary liability)
- Number of automated tests

Moreover, the teams are using an issue tracking and project management software called Jira to manage the work in each sprint, which enables the agile approach in global teams. Jira is also the software that keeps track of the data that constitutes the KPIs and the central source of information regarding tasks in the development process. Employees can move the tasks to other development phases when completed in order to notify each other the current state of the task. For example, a business analyst may have specified requirements and then move the task over to a developer who codes a function and in turn, move the task to QA resources for testing. Moreover, each task is classified when registered in the system based on the origin of the task (bug, change request, customer story etc.).

Based on observations made by the authors, quality is mostly associated with technical aspects while lead-time and continuous improvements of the process are somewhat neglected by most employees. However, concepts like Continuous Deployment and Continuous Integration are well known among the teams, but once again, the focus is on technical aspects. In order to trigger a process thinking in the teams, the manager of Integration & BI have recently introduced six principles that the teams should comply with: *Eliminate waste, Continuous Improvement, Quality starts with me, Continuous Delivery, Establish a normal situation* and *Optimizing the Solution before optimizing the part*. Some principles are more or less integrated into some teams, but overall the principles seem to be hard to comprehend and therefore hard to adapt into the employees everyday working chores.

One of Poppendieck & Poppendieck's (2006) waste categories is *Defects*. Integration & BI had a high understanding regarding this waste and a fully functional approach in order to manage *Defects* in their development. Moreover, the authors considered the technical aspect of this waste too high. Thereof the limited resources of this study were better to use on the remaining wastes which the case company had lower knowledge of. Consequently, the waste category *Defects* was omitted from the study.

5 Waste identification and analysis

A summary of the waste identified in each team is presented in this chapter. Following each team's summary, the identified waste was structured in a table for each respective team for better visualization of the team's situation. Lastly, a root cause analysis of each teams' waste is presented.

The tables illustrate the level of obstacle every waste was considered by the team. The different levels of the obstacle are classified as: (*) The waste exist, but is rare and not considered as a big issue or have a minor impact when occurring; (**) The waste exist and is considered as one of the main obstacles, are frequent or have a big impact. Additionally, a third option was included: (-) meaning that the waste was not identified in the team. Lastly, an analysis of the wastes root causes is presented for each team except in the section of the Integration team, the BOD teams in Sweden and the Philippines who shares a combined analysis. The wastes of these teams were connected in such a way that separate analysis of each team may have led to confusion for the reader.

5.1 IEC

The IEC team have an agile approach to software development which has been used the past two years. Their two-week sprints are split up into two separate parts where the first week is spent by developers coding while QA prepares for testing. During the second week, QA tests the build while the development resources are planning and authoring requirements for the next sprint. This disables QA from being part of the planning process that can be considered heavily development driven, without consideration of QA's capacity. Moreover, co-authorship of requirements by both development and QA are preferred, but not viable in their current approach. Furthermore, QA colleagues are managed by a separate team from development which makes them even more loosely integrated.

The IEC team could identify all the waste categories in their development process although the most problematic waste was *Partially done work*. The QA can be seen as a bottleneck in the development process of IEC, where *Partially done work* is accumulating and sometimes spanning about three sprints back which creates an overload situation. The workload put on QA disables them from catching up and limits the QA's ability to develop new automated tests at the rate desired. Moreover, the shortage of automated tests and the amount of *Partially done work* disables additional and high priority work outside the planning from being part of a release. The risk of not having a stable build at the sprint end is too high which in turn indicates the low flexibility of the process. Although *Partially done work* is a prominent problem, the team has no visualization of the systems *Partially done work* that can indicate an overloaded system.

Partially done work can also be generated by *Delays* on work items when either a pending decision or information is needed from internal or external resources. In the IEC team, *Delays* are a common problem where information needed from internal sources are seen as harmful as external, mainly because of development and QA's mismatched workload. When development plan and design requirements, input from the busy QA resources are needed and when QA plan input from busy development resources are needed which makes up for a frustrating working environment. The IEC team have created an on hold status on work items and these are on a regular basis found in the backlog as flow over from previous sprints. Moreover, it is common that on hold items go back and forth from different colleagues or development phases. This is extra problematic, due to the tight time window where working items must be completed in order to be included in the build and release.

One issue regularly brought up and discussed by the IEC team is problematic *Handoffs* in the development process. The resources are too busy in order to take time and satisfyingly hand over work from one phase to another and considered by the IEC team as a lack of communication, especially between development and QA. Regarding *Relearning*, the IEC team experienced that even if they captured information from previous development, it was very hard to find the demanded data and it was often outdated. Moreover, information captured was regarding things that went wrong and not about successful work and how to execute specific tasks, meaning that they lost valuable information.

Extra features and *Task switching* did occur in the team's software development. However, compared to the other waste classifications considered as a minor problem. The waste identified in the IEC team, together with the respective level of the obstacle is presented in Table 7.

Table 7: Waste identified in the IEC team.

Waste identified – IEC

Waste	Obstacle
Partially done work	**
Extra features	*
Relearning	**
Handoffs	**
Task switching	*
Delays	**

Root cause analysis of waste – IEC

Many of the IEC team's wastes may be originated from the amount of *Partially done work* in their development process. This is highlighted by Poppendieck & Cusumano (2012) who means that this is due to large batches of *Partially done work* created between sequential development processes or in the boundaries between functions, which is consistent with findings in the IEC team. The work accumulating in QA is causing overload situations and in turn, decreases flow which is coherent with the argument made by Petersen & Wholin (2010). Moreover, the overload situation disables the QA team to participate in the planning phase and co-author requirements, which results in problematic *Handoffs* and makes the situation increasingly unwieldy.

Petersen, Wohlin, & Baca (2009) means that high levels of *Partially done work* can result in *Delays*. This was also found in the IEC team where *Delays* were common and harmful where the needed input was delayed from occupied sources.

Morgan (1998) mentions that *Partially done work* causes stress in the organization which may be a reason why the IEC team struggles with *Relearning*. In their current working environment it is hard to find the time to document and consequently, the team cannot rapidly find the right information regarding their development in their documentation. The IEC team often relies upon getting the information personally which introduces, even more delay that could be avoided if knowledge was captured better in their documentation. The focus should

not be on “what not to do” but rather “what to do” in order to increase flow and avoid unnecessary waiting in order to get a response by email.

The root cause of the IEC team’s unsatisfactory *Handoffs*, *Delays*, poor flow and to some degree *Relearning* appears to be connected with batches of *Partially done work*. In order to overcome this problem, the IEC team understands the importance of involving QA in the planning process while also put resources on developing automatic tests. This would reduce the stress of the QA-team while also increase the flexibility of adding high importance work items later to the build. However, the team needs improved approaches to address *Partially done work* and flow of the process as well as improved planning with regards to their capacity.

Petersen & Wholin (2010) highlighted the importance of a continuous flow of work items in software development and that indicators should enable analysis of *Handoffs* and *Partially done work* in order to improve the process flow. The IEC team have problems with both these wastes, but no visualization or measure to indicate the development process current status. Arguably, the KPI *commit to done,* used centrally to evaluate the teams’ performance, measures the inverse of *Partially done work* and can consequently fairly well reveal how much work that is still in the development process. However, this measure will not indicate why or when *Partially done work* is accumulating and causing overload situations and further only recognize the status at the end of the sprint. More or less, it may indicate symptoms of problems but not provide further guidance to root cause analysis.

5.2 BI

During the interview with the BI team, some waste could be identified, while most were considered uncommon. The team work with predictable outcomes and have not missed a single essential deadline during the last four years. *Partially done work* was kept to a minimum by their well-designed development process while *Extra features*, as well as problematic *Handoffs*, practically did not exist. *Task switching* was seldom a problem as their methodology restricted software engineers to work on similar tasks like scripts or reports with only one task at a time. Moreover, *Delays* were a small problem for the BI team since they are relatively isolated in their development from other teams.

The only waste identified that could be considered as an obstacle was a risk rather than an occurring problem. It was notified that some employees hold key competency in certain areas which may lead to problems if those resources are absent or quit. The waste identified in the BI team, together with the respective level of the obstacle is presented in Table 8.

Table 8: Waste identified in the BI team

Waste identified – BI

Waste	Obstacle
Partially done work	*
Extra features	-
Relearning	**(risk)
Handoffs	-
Task switching	*
Delays	*

Root cause analysis of waste –BI

Without a missed deadline during four years, it is clear that the BI team operates predictable with minimized waste in their development process. The success of the BI team is partly the result of having the opportunity to redesign their development process, which was done during a longer period of time when development was at a standstill, and an isolated environment where they operate. The standstill enabled thoroughly consideration of their development process while their lack of dependencies to other teams minimizes long lead-times. Moreover, great process thinking and improvements considered as everyday chores enables the team not only to improve but also to be proactive.

By having software engineers working in their field of expertise exclusively, the flow can be increased and the *Task switching* can be kept at a minimum. However, this can result in problems in the end if the team loses their key competency, and should thus be avoided by expanding the knowledge of all colleagues in order for the risk to be minimized.

5.3 Integration

The Integration team could identify themselves with most of the wastes and described the given material as "recorded reality", where *Delays* were concerned as the major issues. *Delays* were generally the result of dependencies to other teams like the BOD team or teams in the organization developing the software integrating with Infor M3. According to the integration team, the collaboration between teams was insufficient and most only looked at a micro perspective. This makes it problematic when conducting integration projects that usually requires input from numerous different resources. Occasionally *Delays* from other teams also resulted in *Extra features* as overly complex solutions in order to solve easy problems. These "emergency solutions" was required in order to comply with integration projects deadlines. Moreover, this resulted in later backtracking when the preferred solution to the problem was in place. The Integration team was also dependent on the IEC team who builds the tooling used by the Integration team. However, the tooling was considered inadequate by the Integration team and resulted in integrations with lesser quality to the end customer. For example, the integrations were not easily upgraded and subjected to long installation times.

Another waste identified was *Task switching*. The team was required to run multiple projects simultaneously in order to not have too much idle time when frequent *Delays* halt projects from being progressed. In turn, this creates an increased amount of *Partially done work* as open projects, which becomes difficult where many important issues are occurring simultaneously.

The Integration teams work was more of coordinating nature rather than developing which made the wastes *Relearning* and *Handoffs* less applicable. However, to some degree, they occurred, but on a less frequent basis. The waste identified in the Integration team, together with the respective level of the obstacle is presented in Table 9.

Table 9: Waste identified in the Integration team

Waste identified – Integration

Waste	Obstacle
Partially done work	**
Extra features	**
Relearning	*
Handoffs	*
Task switching	**
Delays	**

5.4 BOD (Sweden)

The Swedish BOD team could identify all wastes in their day-to-day business, however at varied levels of relevance. *Delays* and *Task switching* were by the team considered the most harmful wastes and affecting the team's throughput significantly. *Delays* are in most cases the result of dependencies to outside resources as teams managed by the Business Engine unit. In these cases, small fixes may result in significant *Delays* halting the teams progress significantly.

The teams work is almost exclusively controlled by the projects the Integration team run. Since the Integration team is pursuing multiple projects at a time the Swedish BOD team also faces some amounts of *Partially done work*. The PBA role in the BOD team includes both software development as well as coordination tasks which result in substantial *Task switching*. The numerous meetings needing attendance mixed with development tasks that require full focus is heavily impacting the team's productivity. Additionally, the team is generally facing problems with their efforts to truly work with a sprint approach, since unplanned and often critical issues frequently interrupt the progress mid-sprint. Pressure and request come from a number of sources, resulting in situations where the team finds themselves powerless in their own planning. The efforts in planning become obsolete when critical issues arise which results in even more *Task switching*.

Handoffs together with *Relearning* and *Extra features* was considered by the Swedish BOD team as minor wastes. Even though they occurred, the impact was minimized mostly since the team was experienced and had worked with both the product and with the other team members for a long time. The waste identified in the BOD (Sweden) team, together with the respective level of the obstacle is presented in Table 10.

Table 10: Waste identified in the BOD (Sweden) team

Waste identified – BOD (Sweden)

Waste	Obstacle
Partially done work	*
Extra features	*
Relearning	*
Handoffs	*
Task switching	**
Delays	**

5.5 BOD (the Philippines)

The BOD team in Manila shares characteristics with their Swedish counterpart where some wastes are minimized by competence and experience, both as senior engineers and the maturity as a team, leading to smooth *Handoffs* and *Extra features* as an insignificant problem. Beyond *Handoffs* and *Extra features*, waste could be identified in all of the other categories of varied magnitude. For example, *Relearning* was infrequent while *Delays* caused by dependencies was both common and problematic and thus the most disadvantageous waste for the team. Tasks that need changes is outside products may span over multiple sprints since the team needs approval from application owners before the change can be made, even if the change is small.

The team is generally successful in closing most of their planned work items during each sprint and even claimed they had no *Partially done work* at all with 85 % commit to done, something completely contradicting. Moreover, tasks halted by *Delays* are excluded in the sprint backlog and does not affect the teams KPIs, which actually corresponds to more *Partially done work* than perceived. Consequently, *Partially done work* can be considered a main obstacle for the team.

The team faces some *Task switching* and is not able to work on a single project at a time because of the multiple integration projects running simultaneously. However, the team mentions that they would be able to deliver projects with significantly less lead-time if they were able to focus on one projects at a time. The waste identified in the BOD (the Philippines) team, together with the respective level of the obstacle is presented in Table 11.

Table 11: Waste identified in the BOD (the Philippines) team

Waste identified – BOD (the Philippines)

Waste	Obstacle
Partially done work	**
Extra features	-
Relearning	*
Handoffs	-
Task switching	*
Delays	**

Combined root cause analysis of waste – Integration and BOD (Sweden & the Philippines)

One major concern for the Integration team is dependencies to a number of other teams and units: the BOD team and in turn BE, the IEC team and a number of other Infor software in the other end of the integration. However, all these dependencies are acting differently on the Integrations team's ability to run integration projects successfully.

The integration team utilizes tooling developed by the IEC team in order to develop integrations to other Infor software. The tooling in place is considered by the Integration team as insufficient and affects the quality of the complete integration (longer installations and more time-consuming maintenance for the customer). The Integration team is also dependent on BOD development in order to create integrations. However, the Integration & BI unit has recognized this dependency, whereby the two teams Integration and BOD was recently combined together. From a lean perspective, this is a favorable decision since it will provide an improved end to end perspective of the value flow (Petersen & Wholin, 2011).

Infor products on the opposite side of the integration will frequently result in *Delays* for the Integration team for various reasons. For example, some projects have started while the required technology in order to create the finished product is missing in the other product. Hence, *Delays* are introduced where the project is put on hold until the necessary technology is implemented and thus a source of *Partially done work*.

The integration projects are primarily made up by BODs delivered by the BOD team, which hence affects the Integrations team's ability to run projects. In the case of the BOD team, *Delays* connected to dependencies to BE when requesting API changes are their main concern, resulting in substantial *Delays*. Even though API changes are quite small tasks in general, it has to be cleared with the product owner of BE and follow a bureaucratic change request process. This will, in turn, introduce *Delays* for integration projects whereby complex solutions occasionally have to be created in order to complete the integration in time, resulting in an increased technical debt and inefficient usage of the integration team's resources. When at last the API changes are released by BE, the finalized solution can be implemented, but at the cost of even more resources spent that could have been avoided if BE could comply with the integration projects scope.

A number of *Delays* present in the Integration team's projects require multiple projects to be run simultaneously in order to decrease idle time but introduces other wastes like *Partially done work* as projects on hold and *Task switching*. The Integration team state that "The problem with this (multiple projects) is that it will be impossible to plan because it can be quiet one week, but the following week things happen in four directions, all of which are very important. Waiting and multiple tasks or projects maximizes *Task switching* and results in inefficiency and an inability to plan". Running multiple integration projects is challenging, but might be the only solution in order to get any efficient use of the team's resources. However, integration projects conducted simultaneously also results in *Task switching* for the BOD team, something that is extra problematic for the BOD team in Sweden who conducts both coordinating and development tasks. Moreover, the BOD team in the Philippines also recognizes multiple projects run simultaneously as a challenge, although not considered a main obstacle, which creates an increased amount of *Partially done work*. To conclude, dependencies, especially to BE, creates *Delays* for the BOD team and in turn the Integration team. This problem can singlehandedly explain a majority of other waste prominent at both the BOD teams and the Integration team such as *Partially done work* and *Task switching*. Moreover, the dependency to BE also generates technical debt as unjustified complex solutions and consequently both inefficiency and rework.

In order to minimize *Delays* connected to other Infor M3 teams and units as well as other Infor software, the Integration team is striving to only start projects when the scope is concrete and by-in from all respective parties is established. This reduces *Delays* and increases the probability to hold deadlines. From a lean software development perspective, this is an example of applying the lean software development principle Defer commitment, especially advantageous when facing complex problems (Poppendieck & Poppendieck, 2006). However, the above-mentioned principle should be complemented by the lean software development principle deliver fast in order to delay decisions without drawbacks. In the case of the Integration team, this may imply minimization of lead-time as brought up by Petersen (2012). If lead-time in controllable phases can be minimized, the probability of successfully finalize the project in time will increase without the need of inadequate complex solutions. When all information necessary in order to complete the project is in place, a focused endeavor should be able to minimize the other wastes like *Partially done work* and *Task switching*. However, in order to make the right decisions, the awareness of lead-time should be improved, as suggested by Carmel (1995). Currently, the teams have no measures or indicators of the performance in this regard, especially concerning the end to end value flow. Moreover, it was identified that the teams are generally only concerned about the micro perspective which may lead to different agendas within the teams.

6 Waste elimination approaches at Integration & BI

This chapter covers Integration & BI's current approaches regarding reducing waste based on the aspects identified in literature and an analysis of suggestions on how to improve their waste management.

Awareness

The teams were generally lacking knowledge regarding the concept of waste. This can be justified by the novelty of the *Eliminate waste* principle at the unit, which resulted in less successful approaches to reduce waste. Without the necessary knowledge regarding waste, it is difficult to find suitable approaches to minimize it. The low understanding of the concept has probably originated from a lack of a proper definition of waste together with the absence of a common classification model that facilitates the habit of identifying waste in the organization. However, the manager of Integration & BI recognized the importance of the principle and fully supported it, even if the efforts regarding waste was recently initiated.

Indication

Integration & BI was evaluating the teams based on a number of KPIs. These was automatically collected through Jira, but also clear and reliable because of their simple nature as one-dimensional data points at the end of a sprint. However, since the KPIs used by the teams only considered the status of the development at a specific time, they were not providing a foundation for further analysis of waste and their root causes.

The teams lacked adequate measures that indicates process features like flow, lead-time or *Partially done work* and therefore limited the team members' abilities to analyze waste. For example, the inverse of Integration & BI's KPI Commit to done is the flow over of work items to the next sprint and thus indicates a subset of the teams *Partially done work*. However, the KPI did not consider what the level of *Partially done work* has been during the iteration. In order to reach an arbitrary Commit to done the teams' may have been required to work overtime since the flow of the process has been insufficient. The teams' reaches their goal which is perceived positive, meanwhile, the overtime could have been avoided with the right actions to increase the process flow. Commit to done is supposed to measure the teams' ability to plan according to capacity. While the KPI does indicate a potential overestimation of capacity, it will not further facilitate identification of bottlenecks or overload situations and their root causes. In order to identify what actually happened during sprints, data points should be visualized throughout the sprint, an approach which Integration & BI was not practicing regarding any of their KPIs.

In regards to quality Integration & BI have multiple KPIs that indicates the process ability to produce defect-free software. Moreover, since Integration & BI also measures the number of automated tests of each team they have a progressive approach towards measuring quality.

Analysis

Multiple interviewees admitted the lack of structure of sprint reviews as an issue when analysis of problems is performed. The analysis was not predefined in either structure or topic which resulted in somewhat free discussions with a varied outcome. Moreover, mainly technical aspects of the process were discussed while general lean aspects for example flow or lead-time was neglected in the analysis. The analysis can be considered quite uncomprehensive where problems and solutions effect on other process aspects was

overlooked. This way of focusing on the micro perspective during sprint reviews may lead to changes rather than improvements where learning potentials are lost.

The authors observed a dominance of what to avoid rather than root cause analysis. What to avoid can be favored since it is easier and less time consuming to implement in practice, which supposedly can be the result of not allocating resources for improvement work.

Elimination

Integration & BIs teams had some established practices that reduced waste, but the successfulness of applying them was varied. One team that stood out was the BI team that had developed procedures for minimizing most of the waste in their software development. For example, Proof of Concept (POC) was utilized when developing new features. By starting with a POC on a small feature set, before the actual development of the entity, the team could minimize multiple risks. For example, potential *Extra features* with corresponding *Partially done work* can be considered eliminated since it can be managed before full implementation. Consequently, POC implies the minor waste instead of potential major waste.

Another practice used by both the BI team and the BOD teams were peer-review. By peer reviewing when developing code, misunderstandings could be decreased, minimizing problematic *Handoffs*. Moreover, by introducing product managers at an early stage quick feedback could be obtained. This ensured that the features developed are requested by the customer, resulting in the reduction of *Extra features*.

By utilizing standards and documentation the BI team work towards very predictable outcomes with smooth workflow. Moreover, these standards and documentation have led to not a single missed essential deadline during the last four years. A part of this documentation was something called a cookbook: a collection of guidelines and instructions, updated on a daily basis and considered as a part of everyday chores. This makes it easy for the developers to follow certain standards. Additionally, this documentation minimizes the *Relearning* since the employees have the possibility to look at earlier made entities similar to the work items developed. The BI team is an example of Reflective practices – reflecting on action and may be one of the reasons behind the team's successful results.

The BOD teams have also constructed a cookbook, but since it has not been continuously updated for years, it was practically obsolete compared to the one used by the BI. The IEC team, on the other hand, did not have a cookbook or a systematic approach regarding storing information. In their current situation, they used a system that was easy to access, however, requested information was problematic to find when using it. The system was occasionally updated and when it was, it was only regarding items that had gone wrong, making the information stored incomplete from a reuse and learning perspective.

All of the Integration & BI's teams have a sprint approach to software development, which starts with sprint planning and ends with a sprint review. During sprint planning, tasks are prioritized for the upcoming sprint. These prioritizing sessions thereby map out exactly what should be done each sprint, which subsequently leads to decreased amount of *Extra features*. During the sprint reviews problems were analyzed as mentioned before, however, when actions were decided by the teams, it was not considered a work-item as regular work since improvement efforts seldom obtained a Jira ticket and thus planning were not considering this endeavor. Moreover, unstructured and inconsistent follow-ups were observed with no evaluation if the improvement effort resulted in the benefits predicted.

A summary of identified approaches used by Integration & BI is presented in Table 12. The approaches are further structured based on aspect the approach regards.

Table 12: Summary of identified approaches used by Integration & BI

Aspect	Current approach
Awareness	Low understanding of the concept waste
	No classification or definition of waste
	Senior management support
Indication	Clear, reliable and automatically collectable measures
	Shallow, one-dimensional that does not provide means for further analysis
	Not visualized
	Subset of Partially done work indicated with some regards to capacity
	No measures or indicators of flow
	No measures or indicators of lead-time
Analysis	Avoidance of problems. Easy solutions
	Not analyzed combined
	Comprehensive quality measures
	Generally micro perspective and technical aspects of the process
Elimination	POC and Peer review
	Polices and standards at varied degrees continuously improved
	Iterative approach to improvements
	Loosely structured improvements and follow-ups
	Cookbook facilitates reflection on action.

6.1 Analysis - Suggested waste reduction approaches

Table 12 from chapter 6 together with Table 6 from chapter 3.6 were analyzed using a pattern matching method in order to study how Integration & BI's waste reduction approaches could be improved. This resulted in the following chapter. A structured summary of suggested waste reduction approaches together with waste elimination approaches found in literature and in Integration & BI is presented in Appendix B.

Awareness

The support of the manager at Integration & BI is a prerequisite of implementing the *Eliminate waste* principle to its fullest potential. However, the support of the employees is

also important according to Al-Baik & Miller (2014) who highlight the significance of successful early initiatives in order to convince the whole organization. Consequently, the current efforts must be chosen carefully and if possible improve the employees work environment in order to attain full support. However, since the understanding of the concept waste was low at Integration & BI the principle must be introduced more comprehensively. This implies a concrete definition of waste together with classifications in order to get a shared perspective throughout the unit. The teams will be unable to successfully eliminate waste if no habit of seeing waste is introduced, as mentioned by Poppendieck & Poppendieck (2006). Since the classification model by Poppendieck & Poppendieck (2006) was applicable at Integration & BI it should be a suitable model for future initiatives. Moreover, increased understanding can also be achieved by teaching sessions.

Indication

Measures and indicators can function as an important waste identification tool as well as a way to concretize improvements as discussed by Swaminathan & Jain (2012) and Staron (2012). Moreover, Bergman & Klefsjö (2013) also recognizes the importance of making decisions based on facts in order to make improvements. Petersen (2012) takes this further by stating that indicators and measures facilitate implementation of lean software development by continuous improvements. However, Petersen & Wholin (2010) means that analysis of measures and indicators enables the organization to identify root causes of problems. The teams in the Integration & BI unit are currently lacking appropriate measures and indicators that facilitate analysis in order to discover waste and their root causes. Consequently, the introduction of more sophisticated measures could improve the team's ability to eliminate waste. The teams were automatically collecting data from Jira, which makes their current measures both effortless to collect and reliable, which is advantageous according to Staron (2012). They should, therefore, continue to use Jira as far as possible when collecting data for measures and indicators. Jira also facilitates visualization of measures and indicators which the teams should utilize to a greater extent together with a combined analysis of multiple measures (Petersen, 2012).

In the case of the IEC team, much of the identified waste was connected to large batches of *Partially done work* in the system. This is consistent with conclusions by Petersen & Wholin (2010) who argues that high levels of *Partially done work* indicate waste and an absence of a lean development process. Hence, indicators that help the team understand their *Partially done work* during sprints together with measures that can display if development is conducted in small and continuous increments, in other words, flow, would be beneficial.

Petersen & Wholin (2010) suggests that cumulative flow diagram is preferred in order to analyze the development process flow. Cumulative flow diagrams visualize *Partially done work* in specific development phases as well as the nature of *Handoffs*. Moreover, it enables identification of bottlenecks and thus works as a foundation for continuous improvements (Petersen & Wholin, 2011). Consequently, Cumulative flow diagrams would be an advantageous approach first and foremost for the IEC team, but also for other teams conducting development tasks. This in order to facilitate both identification and elimination of waste in their process by root cause analysis. Cumulative flow diagrams were also supported by practitioners earlier studies (Petersen & Wholin, 2010; Petersen & Wholin, 2011; Petersen, 2012) since it was quick to implement.

The level of *Partially done work* of the IEC team should be considered in relation to capacity in order to avoid overload situations, as highlighted by Petersen & Wholin (2010). Even though Cumulative flow chart will give some guidance in this endeavor, statistic control charts can further enhance the ability to detect when too much *Partially done work* is pushed

into the development process. Petersen & Wholin (2010) means that statistic process charts of *Partially done work* levels provides a good basis for further discussion of capacity, something that should be advantageous for the team.

The Integration team together with the BOD teams in Sweden and the Philippines waste was connected to dependencies that caused significant *Delays* and occasionally *Extra features*. Moreover, this resulted in the necessity of running multiple integration projects simultaneously which introduced *Partially done work* and *Task switching*. The *Delays* caused by the dependencies may be enhanced since the teams are generally concerned about the micro perspective. Further, if lead-time in controllable phases can be minimized, the probability of successfully finalize the project in time will increase. Since the Integration & BI unit are currently not able to run all integration projects demanded, increased throughput would be beneficial, which further supports the need for improving the end to end value stream.

In order to improve the lead-time of integration projects, the awareness of lead-time should be improved, as suggested by Carmel (1995). Moreover, when distinguishing between value adding and waiting time, improvement efforts can be indicated (Petersen, 2012). VSM is one practice that facilitates understanding of workflows with an end-to-end perspective and activates practitioners from entire value stream. Consequently, VSM should be a suitable tool in order to identify improvement efforts and create an end-to-end perspective of Integration projects with practitioners from the Integration team, the BOD teams, as well as BE teams.

Analysis and elimination

All the teams in Integration & BI were using an iterative approach for improvements in their sprint review sessions which are a great foundation for waste reduction. However, the lessons learned was relatively ill-structured which often resulted in shallow analysis of problems and led to an elimination of symptoms rather than problems. Moreover, when improvements were decided the structure of planning and follow-ups can be considered inadequate which further limit the team's ability to make significant improvements.

In order to more comprehensively analyze problems and more systematically eliminate waste, the teams could benefit from Deming's (1993) PDSA-methodology covering activities throughout sprints. This would provide a systematic and continuous basis for waste reduction and improvements in general. Since the teams already had an iterative approach for improvement efforts, The PDSA-methodology would only be functioning as an extension of the already established practices.

The sprint reviews may serve as the planning phase where waste could be identified by analyzing indicators. This analysis should be based on multiple indicators and quality measures in order to gain a more comprehensive understanding of the problem and introduce increased systems thinking in the teams. However, root causes need to be identified in order to prevent waste from occurring again. One approach to identifying root causes is 5 whys mentioned by Miller & Al-Baik (2016). 5 whys can be used both during sprint review, but also during daily work by the employees. This would facilitate that improvement efforts are actually eliminating waste instead of just removing the symptom. Moreover, the analysis should consider an end-to-end perspective of the value flow, emphasized by Poppendieck & Cusumano (2012) and Petersen & Wholin (2010), in order to introduce a broader overview of waste and its affect outside the teams. Consequently, the teams should question proposed improvement suggestions and their affect outside the team before making decisions. Lastly, when an improvement effort is decided, the result from the effort should be predicted in order to facilitate validation at a later stage.

The introduction of a more refined analysis of problems may enable the teams to understand the importance of waste reduction efforts and thus ensure higher prioritization of such endeavors. However, waste reduction efforts should be planned with Jira tickets so an avoidance of performing that task will affect the teams other KPIs. This would further ensure that the teams are not planning over capacity or just ignore improvement efforts without consequences.

Policies & Standards and Double loop learning can serve as sound elements in a well-functioning PDSA-methodology ensuring improvements are performed systematically (the *Do* phase) and followed-up as well as questioned by its validity (*Study* phase). This can be achieved by evaluating performed improvement efforts and validate the result by analysis at the end of a sprint, hopefully, indicated by measures established by the team. If the result is satisfactory it should become standard practice while unexpected outcomes may lead to a succeeding PDSA-loop in the next sprint in order to ensure that failed improvement efforts are not neglected, as proposed by Deming (1993).

POC and peer review are great examples of practices to reduce waste in software development. However, they are specific practices targeting specific waste, thus even if they reduce the waste they may be the result of eliminating symptoms rather than the real problem. In cases like this, the concept Double loop learning highlighted by Miller & Al-Baik (2016) may provide a fruitful addition. Since the practices POC and peer review are tasks that spend resources, disregarding how well functioning they may be, they should be questioned of why they are performed in the first place and if they are currently needed. By always questioning tasks the teams can ensure that established practices are not executed in excess and more importantly understand when they are the most needed.

The cookbook is an another tool for achieving continuous improvements if managed correctly. The use of standards that BI could exhibit, is a prime example of how standards should work according to Kondo (2000). Furthermore, their approach towards the cookbook can be linked to the self-determined standard by Miller & Al-Baik (2016), since the employees themselves update it on a regular basis. The cookbook also serves as a *reflection on action* from the Reflective practice (Miller & Al-Baik 2016) since the cookbook forms a single source of easily accessible past experiences. Therefore, the other teams should monitor and learn from the BI team on how they have so successfully introduced standards and Reflective practice in their team. However, since the context is different in the teams, practices should not be transferred between the teams without adaption as highlighted by Poppendieck & Poppendieck (2003) and Al-Baik & Miller (2014). Instead, the BI team's practices should be a source of inspiration that facilitates organic growth of practices within the other teams, that may or may not include elements of the BI team's practices.

7 Conclusion & Recommendations

In this chapter, the conclusions of the thesis are presented. In order to make the conclusions orderly and comprehensible, the chapter is divided into different sections, all of which answers one of the study questions of this study. Lastly, the recommendations regarding waste elimination approaches at the Integration & BI unit will also be presented.

7.1 Conclusions

The aim of this thesis was to identify waste and suggest approaches to reduce waste at the unit Integration & BI at Infor M3. In order to accomplish this, three study questions were formulated, all of which are answered in this chapter.

- **SQ1**: What is considered waste in software development?

After the conducted literature review, several different classifications considering waste in software development was discovered. The common denominator amongst all the various literature was that the researchers shared the same view of waste. Waste was identified as activities that do not contribute to any sort of value to the customers. Hence, the concept waste in Lean software development is defined alike even if the view of how waste manifests itself may be a disputed subject.

Korkola & Maurer (2014) conducted a study focused on the identification of communication waste within globally distributed software development teams. However, their study did not cover software development on a general basis which limits its applicability. Only four classification models of waste were discovered regarding waste on a general basis in software development. The researchers behind these classifications were Al-Baik & Miller (2014), Poppendieck & Poppendieck (2003, 2006) and Mandić et al. (2010) which are presented in Table 4.

Mandić et al. (2010) identified Ohno's (1988) original seven wastes of manufacturing in a software development context and added new sources of waste regarding decision making. These was somewhat contradicting to Poppendieck & Poppendieck's (2003, 2006) work. However, the fact that Ohno's (1998) wastes were applicable in a software development context strengthens Womack, Jones, & Roos (1990) suggestion that lean philosophy is relevant in any industry.

The classification model of waste by Poppendieck & Poppendieck in 2003 differs from the one presented in 2006. Furthermore, Al-Baik & Miller (2014) based their study on a single case company, meaning the generalizability is low. However, Al-Baik & Miller (2014) was the only study found that did not originate from a manufacturing industry. Consequently, it can be concluded that the subject is still in an early stage and that development within the subject is still occurring. To the best of our knowledge, the only deductive studies applying classification models were by Mujtba, Feldt, & Petersen (2010) and Ikonen et al. (2010). Both these studies applied classification models created by Poppendieck & Poppendieck (2003, 2006) supporting their generalization potential.

Poppendieck and Poppendieck (2006) argues that the classification by its own is not important, but rather the mindset that evolves from the usage of it. By creating an understating regarding waste, it is possible to attain a habit of seeing waste. Moreover, identification of waste should not be perceived as negative, and no organization is entirely free from waste.

- **SQ2**: What is Integration & BI's current waste situation?

All waste categories described by Poppendieck & Poppendieck (2006) could be identified in all of Integration & BI's teams with the exception of the BI & BOD teams that could not identify the wastes *Handoffs* and *Task switching*. The classifications of waste by Poppendieck & Poppendieck (2006) can, therefore, be concluded adequate in the case study at Integration & BI. Additionally, the argument regarding using their classifications in order to attain a mindset is reinforced since the waste is described differently in the teams, even when describing the same category of waste.

Evaluating the most prominent waste at each team can introduce solutions for improvements and further give guidance in where to start when conducting root cause analysis. In the case of the IEC team, accumulating *Partially done work*, unsatisfying *Handoffs*, regular *Delays* and consistent *Relearning* was considered most problematic while the Integration team was subjected to *Delays* from multiple sources, *Partially done work* in numerous projects and frequent *Task switching*. Moreover, the BOD team suffered from significant *Delays* from BE, further, the authors observed *Partially done work* in the Philippines team while their Swedish counterpart mainly faced *Task switching* as the most prominent problem. Lastly, it was difficult to classify any waste in the BI team as problematic, but instead, waste was brought up more or less as a risk rather than a current problem. The BI team did, however, agree with the wastes defined by Poppendieck & Poppendieck (2006) and mentioned: "They are a great reminder and something you should consider on a regular basis".

The root cause of most of the IEC team's wastes can be derived from the large batches of *Partially done work* created between development and QA which is consistent with arguments made by Poppendieck & Cusumano (2012). The amount of *Partially done work* creates overload situations and decreases flow, hindering their QA employees of participating in the planning phase of sprints which in turn increases the challenge of *Handoffs*. Petersen, Wohlin, & Baca (2009) findings are consistent with findings in the IEC team where *Delays* were both common and harmful since required input was delayed from occupied sources. Morgan (1998) mentions that *Partially done work* causes stress in the organization which may be a reason why the IEC team struggles with *Relearning*. It was hard for the IEC to find the time to document consistently and consequently, the team could not rapidly find the right information regarding their development in their documentation.

Dependencies are one main concern for the Integration team resulting in *Delays*. Since the *Delays* are so substantial multiple projects are required to be run simultaneously, which increases *Partially done work*. Moreover, this makes planning in many cases obsolete where important issues are introduced in multiple projects at once, which creates *Task switching*. Though *Delays* as a result of dependencies may be the root cause of the team's difficulties, the situation is expected to improve since they joined the BOD team.

In the case of the BOD team, *Delays* due to dependencies to BE when requesting API changes are the most prominent struggle in turn affecting the Integration team. These *Delays* commonly introduce *Partially done work* in the team with tasks that can not be completed for a significant amount of time. When the Integration team runs multiple projects, as mentioned above, the BOD team is also affected and results in *Task switching* at a level that it sometimes stops production completely. This problem can partly be originated from *Delays* of BE that affects the whole value chain and thus the Integration teams commitments in multiple projects.

The BI team's most current waste *Relearning* was more of a risk and not considered a waste by now. Developers work in their field of expertise exclusively and the *Task switching* can be kept at a minimum but may result in loss of key competency.

It could also be concluded during the study that it is difficult to quantify the potential savings from the reduction of waste. However, Al-Baik & Miller (2014) could show a 55% improved lead-time from reducing waste at a software provider. This example indicates that the reduction of waste in software development is beneficial. The authors think that the difficulties regarding the quantification of waste are based on the dependencies between the wastes and the associated complexity. Further, all of the interviewees agreed that the wastes brought up during the interviews were considered as the main obstacles to their current development process. Thereby the productivity should increase if these obstacles were reduced. Lastly, in order to reach Infor M3's mission to deliver software as fast as possible, decreasing the lead time is vital, where the reduction of waste has been proven as a successful method by Al-Baik & Miller (2014).

- **SQ3**: How can Integration & BI reduce waste in their software development?

The support of the *Eliminate waste* principle should be further attained by the employees by focusing on making early endeavors significant. The unit further needs a concrete definition of waste and a classification model as a thinking tool, where Poppendieck & Poppendieck's (2006) work should be valid. The understanding of the concept should also be increased for the employees, reinforcing a habit of seeing waste.

POC and peer review are established practices that reduces waste. Such practices that targets specific waste should continually be used but also questioned, as the concept Double loop learning by Miller & Al-Baik (2016) suggests, in order to ensure that established practices are used as appropriate as possible.

The use of standards by the BI team is a prime example of how they should be used. Their continuous updates of their cookbook are executed by the users of it is concurrent with recommendations of Miller & Al-Baik (2016). The other teams should monitor and learn from the BI team regarding how they have so successfully introduced standards and elements of *Reflective practice* in their team.

All Integration & BI teams were using an iterative approach for improvements and waste reduction during sprints while the activities were somewhat poorly structured. The teams should be benefited from a more systematic approach that a PDSA-methodology will offer. However, root causes need to be identified in order to prevent waste from occurring again. One approach to identifying root causes is *5 whys* mentioned by Miller & Al-Baik (2016) which would be well suited as an activity in the PDSA-methodology. Moreover, when performing improvement efforts, the end-to-end perspective of the value flow should be considered. Consequently, the teams should question proposed improvement suggestions and their affect outside the team before making decisions. Policies & Standards and Double loop learning can also serve as sound elements in a well-functioning PDSA-methodology ensuring improvements are performed systematically and followed-up as well as questioned by its validity.

Integration & BI was missing measures and indicators comprehensive enough in order to facilitate analysis of waste and their root causes. Consequently, a number of new measures and indicators should improve their waste elimination approach. While new measures may be introduced, they should continue to utilize Jira for automatically collecting data but also exploit Jira's ability to visualize the data.

Cumulative flow diagrams can help visualize *Partially done work* and *Handoffs* in development processes and could highly improve the IEC team's ability to analyze their current situation. Moreover, it has been shown in earlier studies by Petersen & Wholin (2010, 2011) and Petersen (2012) that the practice is easy to implement. Moreover, *Partially done work* should be considered in relation to capacity and monitored by using SPC which provides a basis for further discussion of adequate workload.

The Integration team together with the BOD teams should analyze lead-time of integration projects in an end to end perspective. If lead-time in controllable phases can be minimized, the probability of completing the project in time will increase and possibly increase the capacity of running integration projects. VSM should be a suitable practice in order to identify improvement efforts and create an end-to-end perspective of Integration projects with practitioners from the Integration team, the BOD teams, as well as BE teams.

7.2 Recommendations

The result of the study was presented to Integration & BI unit May 5th, 2017 where all teams were represented. They approved to the suggestions of how the unit could improve their waste elimination approaches, whereby investigation of early implementation was decided. Cumulative flow diagram and continuous improvements through a PDSA-methodology during sprint reviews was decided as the first practices to be tested. Whereby the prerequisites and applicability of Cumulative flow diagram in the IEC team together with a draft of a PDSA practice tested by the BOD team in the Philippines were initiated.

The issue tracking and management software Jira, that all Integration & BI teams already use, has the functionality to visualize sprints as cumulative flow charts, thus the technical aspects of implementing the diagram into IEC's current work practices is minimal. It is more the matter of increased knowledge of flow and how to identify waste in the diagram that is the challenge for successful implementation. Thus, Cumulative flow diagrams will be used by the team in their upcoming sprints in order to analyze the process flow and identify potential improvements. Moreover, a draft PDSA-methodology with root cause analysis that will be tested by the BOD team in the Philippines in order to introduce continuous improvements in their development process will be introduced in the upcoming weeks.

Following the use of Cumulative flow diagram in IEC and PDSA in BOD (the Philippines), the first step in order to improve waste elimination at Integration & BI the concept of waste must be introduced to all the unit's coworkers. This is mandatory in order to make waste identification and elimination a habit in the unit. Moreover, if employees understand the benefit of waste elimination and how it can improve their work environment the implementation would be more likely to attach to the organization. The BI team was successful in their current waste elimination approach regarding standards, especially their efforts regarding the cookbook. The other teams are recommended to follow BI's example and find out how they can adapt the BI teams' successful approaches to their team's context.

As the next step, the Integration & BI unit is recommended to further make the PDSA-methodology a standard practice in all of their teams in order to attain more structured improvement efforts by root cause analysis and follow-ups. Consequently, the BOD team in the Philippines should share their experiences of using the practice after a few sprints to the other Integration & BI teams. Moreover, Cumulative flow diagrams would be a beneficial practice for every team that mostly have software development responsibilities, like the BOD team in the Philippines, whereby the application of Cumulative flow diagrams should be investigated here as well.

Other measures and indicators could be beneficially used by the Integration & BI unit. For example, statistic control charts based on *Partially done work* is a practice recommended to use by the IEC team in order to facilitate better analysis of their capacity. Firstly, the IEC team is recommended to monitor the level of *Partially done work* in their development process and estimate the mean of partially done work during sprints together with standard deviation. Secondly, the team could visualize the level of partially done work with regards to the mean and control limits of +/- 3 standard deviations. Laslty, the IEC team could use the chart as a foundation for further discussion regarding what level of *Partially done work* is suitable for the team and take actions based on the newly gained information. However, Jira has no current functionality of automatically generating statistic control charts of *Partially done work*. Thus, the implementation of the measure is more difficult since it requires investigation of how the chart could be generated outside of Jira. Consequently, the approach should be strived for, but further investigation of the ability to automatically generate the charts is needed.

Lastly, VSM is a recommended practice in order to identify improvement efforts and create an end-to-end perspective of Integration projects with practitioners from the Integration team, the BOD teams, as well as BE teams. The managers of Integration & BI team were however quite skeptical to the practice since it requires resources from multiple teams that need to be coordinated in a time-consuming activity. The authors recognize the significance of the practice and highly suggest that VSM should be conducted even if the time consumption of the practice is substantial in relation to the other recommendations. The VSM could enable identification of non-value adding time that may been able to be minimized. If so, the lead-time of integration projects could be minimized and thus enable the teams to run less integration projects simultaneously without inadequate idle time. Moreover, the authors strongly believe the Integration, BOD and BE teams could achieve an increased understanding of the value flow by practicing VSM and find multiple process steps that could be improved based on the fact that the teams faces significant delays originated from essentially effortless changes.

8 Discussion

In this chapter, the authors give their subjective view on how the process of the thesis has been conducted. Furthermore, a discussion regarding the validity & reliability is presented to the reader and lastly, suggestions for future studies are given.

8.1 The thesis process

In the beginning of the thesis, the intended purpose of the study was to cover all of the seven principles of lean software development at Integration & BI. However, the chosen area of software development was new to the authors, who had not thought of the complexity in software development. As a result, a couple of weeks was spent just in order to understand the daily work conducted at Integration & BI, whereby an understanding emerged. The authors realized the scope of implementing all of the principles and through this understanding the delimitation to only investigate the principle of *Eliminate waste* was born.

Due to the complexity of the subject, an iterative approach had to be conducted during the study. Since the understanding of the software development was difficult to comprehend for the authors and some of the material given to Integration & BI was misinterpreted. But by utilizing an iterative approach during the time span of the thesis, comprehension was created as well as misinterpretations were eliminated. Moreover, the approach helped the authors to construct the different delimitations and study questions since new information became available continuously.

The first study question, SQ1, served as a crash course in the waste identification and reduction in software development. Further, it was completely answered by the conducted literature review. The second study question, SQ2, constituted the situation analysis at Integration & BI. SQ2 was mainly answered by the empirical findings, however, some outcomes from the literature review were needed since the interview guide was constructed from it. Lastly, the third study question, SQ3, utilized both the empirical findings as well as the literature review in order to be answered. Through this, recommendations could be developed for the daily work at Integration & BI.

Regarding the data collection, the authors first thought that semi-structured interviews with standardized questions would be the right way to go. This was based on the belief that with standardized questions, it would be easy to compare the different teams against each other. However, after the pilot interview, it became clear that non-standardized questions were the best alternative. As a result of different working methods and tasks in this complex environment, standardized questions would not have been beneficial in order to reach a deep understanding regarding waste at Integration & BI. Regarding the selection of the interviewees, the authors are pleased with the given outcome. However, if time and resources would have been unlimited, representatives from the BE team, as well as the Swedish IEC and Philippine Integration team, would have been interviewed as well. These interviews may have revealed more potential wastes and additional dependencies and patterns.

Further, the authors visited the office in Stockholm for a total amount of three weeks. During the time at the office a lot of observations could be done, as well as quick feedback could be received by the employees. Thereof a longer stay at the office may have resulted in a more comprehensive result. However, this would not have helped regarding the Philippine teams or the other offices in Sweden.

If the authors would have changed anything during the thesis, it would have been to include the wastes of Al-Baik & Miller (2014) in the existing framework that was utilized. Because

of the limited time during the study and the changes of delimitations and study questions, only Poppendieck & Poppendieck's wastes was considered when conducting the interviews. Even though these wastes are on a very general basis and covers almost every possible waste, the authors encountered some rare problems that may been more easily classified by some other waste classification. For example, the Integration team had to create a very complex solution as a respond to delays from IEC, which had to be classified as an extra feature. Further, the BI team saw a risk of competence loss, which the authors had to classify as a relearning problem. These examples were not completely consistent with Poppendieck & Poppendieck (2016) categories, even if they were related in aspect, and may have been better categorized in some other waste categorization model. However, the authors are very pleased with the given result and some of the recommendations conducted during the study have already been implemented. The IEC team have implemented the cumulative flow chart and the BOD team in the Philippines have started using a PDSA methodology during their sprints. However, the most important contribution of this study is the establishment of awareness regarding waste at the different teams.

8.2 Validity & Reliability

During the conduction of this thesis, various sources of information have been used, both considering the collection of secondary data as well as primary data and were done in order to strengthen the credibility of the study. Through books, journals and conference notes could information be gathered regarding waste in software development. Further, from interviews and observations, the current situation at Integration & BI could be identified regarding waste. Most of the information gathered from observations was also sent to the Supervisor at Integration & BI, so that he could verify if information regarding their organization had been interpreted correctly. Moreover, a draft presentation was held with all of the respondents to ensure that the conclusions and recommendations were relevant and realistic.

The findings and results in this study were achieved during a limited period of time. This implies that the results may have looked differently if the study would have been done in another time frame. There is a possibility that projects and working tasks, would have resulted in different results than the current ones. However, the majority of the wastes seemed to be on a general basis, with problems occurring through multiple sprints, which thereby contradicts this thought.

In order to ensure that the employees that were interviewed understood the subject, a summary regarding waste in software development was handed out in beforehand. Further, the first question in every interview was if the respondent understood the given material and if they had any questions regarding the material, if this was the case these were clarified before the interview started. Additionally, the interviews were recorded to ensure that misinterpretations would be minimized. However, misinterpretations did occur during the study but were managed by extra communication. Lastly, the authors never faced any resistance from the interviewees during the interviews, but rather benevolence to help regarding the identification of wastes at the unit.

The number of participants in each interview varied. The authors could not identify any trends amongst the different interviews regarding positive or negative answers from the teams with more participants and vice versa. Yet, there is a possibility that the teams with more participants during the interviews had discussed and reflected together before the interview.

To achieve the best possible outcome from the interviews, the supervisor at Integration & BI helped the authors to choose the individuals that attended the interviews. The authors were new to the organization and had little knowledge of the employees and due to the fact that the

teams were spread across the globe. Together with the limited time and resources given during the thesis, this was both the simplest and best solution according to the authors. However, this may have impacted the validity of the study, since the authors were not in full control regarding the sample selection. Lastly, the lack of prominent waste at some teams can also be a result of not digging deep enough during the interview.

Furthermore, it became clear that in the area of lean software development and handling of waste in software development, the practitioners are ahead of the Academy. This results in the utilization of material that is not taken straight from the academy. Moreover, the area is still at an early stage, meaning that only a little information about some fields can be found. In the current state, the waste classification by Poppendieck & Poppendieck (2003, 2006) are the only accepted ones.

This study was conducted at a single case organization, within an area that continuously changes. However, most recommendations in this study will probably be valid for software development in general. Thereof, the authors believe that other actors within the software development industry can take part of this thesis and utilize some parts in order to enhance their waste identification and reduction approaches.

8.3 Suggestions for future studies

As mentioned in the Validity & Reliability chapter, practitioners are pushing the frontier forward on the subject of waste identification and reduction in software development. In today's research, only one study could be found during this thesis that contradicted the seven wastes established by Poppendieck & Poppendieck (2003, 2006), which was made by Al-Baik & Miller (2014). This study was also the only one that did not originate from the manufacturing industry. Furthermore, no meta studies could be found that touched the concerned area. Thereof a suggestion to future studies would be that more inductive research like the study conducted by Al-Baik & Miller (2014) should be done within the area. This in order to discover new classifications of waste in the software development industry, or verify the already existing classifications. Further, deductive studies can be performed on the findings from Al-Baik & Miller (2014), in order to exhibit if these findings are suitable as waste categorization in software development.

Lastly, as mentioned in the conclusion there are no existing studies regarding the potential savings generated by reduction of waste. This study has mainly focused on the identification of waste in software development as well as suitable approaches to minimize these. Currently, studies that show the potential of lead-time reduction by minimizing waste are published, which multiple research indicate as vital for software providers. However, studies that exemplify potential savings from waste reduction would be beneficial for this study, since it could quantify the potential savings from the recommended actions. Therefore, future studies regarding such a framework are sought by the authors.

9 References

Adams, J., Khan, H., & Raeside, R. (2014). *Research Methods for Business and Social Science Students APA (American Psychological Assoc.)* (Vol. 2). New Delhi: Sage Publications.

Al-Baik, O., & Miller, J. (2014). Waste identification and elimination in information technology organizations. *Empir Software Eng, 19*, pp. 2019–2061.

Antosz, K., & Stadnicka, D. (2017). Lean Philosophy Implementation in SMEs – Study Results. *International Conference on Engineering, Project, and Production Management* (pp. 25-32). ELSEVIER.

Argyris, C. (1994). Initiating Change that Perseveres Chris Argyris. *Journal of Public Administration Research and Theory, 4*(3), pp. 343-355.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., . . . Zaharia, M. (2010). A View of Cloud Computing. In *A View of Cloud Computing* (pp. 50-58). Communications of the ACM.

Bergman, B., & Klefsjö, B. (2013). *Kvalitet från behov till användning* (5:e upplagan ed.). Lund: Studentlitteratur AB.

Bhasin, S., & Burcher, P. (2006). Lean viewed as a philosophy. *Journal of Manufacturing Technology Management, 17*(1), pp. 56-72.

Björklund, M., & Paulsson, U. (2012). *Seminarieboken.* Studentlitteratur.

Campbell-Kelly, M. (2001). Not Only Microsoft: The Maturing of the Personal Computer Software Industry, 1982-1995. *The Business History Review, 75*(1), pp. 103-145.

Carmel, E. (1995). Cycle Time in Packaged Software Firms. *Journal of Product Innovation Management, 2*(12), pp. 110-123.

Chou, S.-W., & Chiang, C.-H. (2013). Understanding the formation of software-as-a-service (SaaS) satisfaction from the perspective of service quality. *Decision Support Systems, 56*.

David, M., & Sutton, C. (2016). *Samhällsvetenskaplig metod.* Lund: Studentlitteratur.

Deming, W. E. (1983). *Out of the crisis.* Cambridge university press.

Deming, W. E. (1993). *The New Economics.* USA: Massachusetts Institute of technology.

Demirkan, H., Cheng, H. K., & Bandyopadhyay, S. (2010). Coordination Strategies in an SaaS Supply Chain. *Journal of Management Information Systems, 26*(10).

Dimitrios, Z., & Dimitrios, L. (2012, 12 13). Future Generation Computer Systems. *Department of Product and Systems Design Engineering*.

Freeman, P. (1992). Lean concepts in software engineering. *International Conference on Lean Software Development* (pp. 1-8). Stuttgart: IPSS-Europe .

Goode, S., Lin, C., Tsai, J. C., & Jiang, J. J. (2015). Rethinking the role of security in client satisfaction with Software-as-a-Service (SaaS) providers. *Decision Support Systems, 70*.

Goutas, L., Sutanto, J., & Aldarbesti, H. (2016). The Building Blocks of a Cloud Strategy: Evidence from Three SaaS Providers. *Communications of the ACM, 59*(1).

Hibbs, C., Jewett, S., & Sullivan, M. (2009). *The art of lean software development.* Sebastopol: O'Reilly Media.

Ikonen, M., Kettunen, P., Oza, N., & Abrahamsson, P. (2010). Exploring the Sources of Waste in Kanban Software Development Projects. *36th Euromicro Conference* (pp. 376-381). Software Engineering and Advanced Applications (SEAA).

(2017). *Infor Corporate Fact Sheet.* Infor.

(2016). *Infor corporate overview.* Infor.

Kaltenecker, N., Hess, T., & Huesig, S. (2015, 08 19). Managing potentially disruptive innovations in software companies: Transforming from On-premises to the On-demand. *Journal of Strategic Information Systems*, p. 17.

Khurum, M., Petersen, K., & Gorschek, T. (2014). Extending Value Stream Mapping Through Waste Definition Beyond Customer Perspective. *Journal of Software: Evolution and Process, 26*(12), pp. 1074-1105.

Kim, J., Hong, S., Min, J., & Lee, H. (2011). Antecedents of application service continuance: A synthesis of satisfaction and trust. *Expert Systems with Applications, 38*.

Klefsjö, B., Eliasson, H., Kennerfalk, L., Lundbäck, A., & Sandström, M. (2010). *De sju ledningsverktygen* (Vol. 8). Lund: Studentlitteratur.

Kondo, Y. (2000). Innovation versus standardization. *The TQM Magazine , 12*(1), pp. 6-10.

Korkola, M., & Maurer, F. (2014, 04 08). Waste identification as the means for improving communication inglobally distributed agile software development. *The Journal of Systems and Software*, pp. 122-140.

Lev-Ram, M. (2015, 08 15). *The redemption of charles phillips*. Retrieved from Fortune: http://fortune.com/2013/08/15/the-redemption-of-charles-phillips/

Li, C., & Li, L. (2013, February 7). Efficient resource allocation for optimizing objectives of cloud users, IaaS provider and SaaS provider in cloud environment. *The Journal of Supercomputing*.

Liker, J. (2004). *Becoming Lean - Inside stories of U.S. Manufactures.* Productivity Press.

Liker, J. (2009). *The Toyota Way* (Vol. 1). Liber.

Mandić, V., Oivo, M., Rodríguez, P., Kuvaja, P., Kaikkonen, H., & Turhan, B. (2010). What Is Flowing in Lean Software Development? *Lean Enterprise Software and Systems*, (pp. 72-84).

Marcello, S. (2016, 09 20). *smartsheet*. Retrieved from agile-vs-scrum-vs-waterfall-vs-kanban: https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban

McManus, H., & Millard, R. (2004). *Value stream analysis and mapping for product development.* the International Council of the Aeronautical Sciences.

Middelton, P. (2001). Lean Software Development: Two Case Studies. *Software Quality Journal*, pp. 241-252.

Middelton, P., & Joyce, D. (2012, 02). Lean Software Management: BBC Worldwide Case Study. *IEEE Transactions on Engineering Management, 59*(1).

Miller, J., & Al-Baik, O. (2016). Kaizen Cookbook: The Success Recipe for Continuous Learning and Improvements. *Hawaii International Conference on System Sciences. 49*, pp. 5388-5397. IEEE.

Morgan, T. (1998). *Lean manufacturing techniques applied to software development.* Massachusetts Institute of Technology.

Mujtaba, S., Feldt, R., & Petersen, K. (2010). Waste and Lead Time Reduction in a Software Product Customization Process with Value Stream Maps. *21st Australian Software Engineering Conference (ASWEC)* (pp. 139-148). Auckland: IEEE.

Murugaiah, U., Benjamin, S., Marathamuthu, M., & Muthaiyah, S. (2010). Scrap loss reduction using the 5-whys analysis. *International Journal of Quality & Reliability Management, 27*(5), pp. 527-540.

Náplava, P. (2016). Evaluation of Cloud Computing Hidden Benefits by Using Real Options Analysis. *ACTA INFORMATICA PRAGENSIA Acta Informatica Pragensia, 05*(02), pp. 162–179.

Naone, E. (2009, July/August). Conjuring Clouds. *Technology Overview*.

Ohno, T. (1988). *Toyota Production system- Beyond Large-Scale Production.* Productiviy Press.

Petersen, K. (2010). An Empirical Study of Lead-Times in Incremental and Agile Software Development. *International Conference on Software Process (ICSP 2010)* (pp. 345-356). Paderborn: Springer.

Petersen, K. (2012). A Palette of Lean Indicators to Detect Waste in Software Maintenance: A Case Study. *Agile Processes in Software Engineering and Extreme Programming* (pp. 108-122). Malmö: Springer.

Petersen, K., & Wohlin, C. (2010). Software process improvement through the Lean Measurement (SPI-LEAM) method. *The Journal of Systems and Software, 83*(7), pp. 1275–1287.

Petersen, K., & Wohlin, C. (2011). Measuring the Flow in Lean Software Development. *Software Practice and Experience, 41*(9), pp. 975-996.

Petersen, K., Wohlin, C., & Baca, D. (2009). The waterfall model in large-scale development. *International Conference on Product-Focused Software Process Improvement* (pp. 386-400). Berlin: Springer.

Poppendieck, M., & Cusumano, M. A. (2012). Lean software development: A tutorial. *IEEE software, 5*(29), pp. 26-32.

Poppendieck, M., & Poppendieck, T. (2003). *Lean Software Development.* Boston, Mass.: Addison-Wesley.

Poppendieck, M., & Poppendieck, T. (2006). *Implementing lean software development: from concept to cash.* Boston: Pearson Education, Inc.

Rodríguez, P., Partanen, J., Kuvaja, P., & Oivo, M. (2014). Combining lean thinking and agile methods for software development: A case study of a finnish provider of wireless embedded systems detailed. *47th Hawaii International Conference* (pp. 4770-4790). System Sciences (HICSS).

Russell, B. (2003, 05/06). Delivery Excellence: A Practical Program for Continuous Improvement. *IT Proffesional Magazine, 5*(3), pp. 37-41.

Samrajesh, M., Gopalan, N., & Suresh, S. (2016). A scalable component model for multi-tenant SaaS application. *Int. J. Advanced Intelligence Paradigms, 8*(2).

Saunders, M., Lewis, P., & Thornhill, A. (2009). *Research methods for business students.* Pearson Education Limited.

Saunders, M., Lewis, P., & Thronhill, A. (2016). *Research Methods For Business Students* (Vol. 7). Pearson.

Saurabh, S., Young, S., & Jong, H. (2016, 09 01). A surveyoncloudcomputingsecurity:Issues,threats,andsolutions. *Journal ofNetworkandComputerApplications*, pp. 200-222.

Sinnett , W. M. (2010). *Software as a Service:Experiences of SMBs.* Financial Executives Research Foundation.

Staron, M. (2012, 02 21). Critical role of measures in decision processes: Managerial and technical measures in the context of large software development organizations. *Information and Software Technology, 54*(8), pp. 887-899.

Stoica, M., Mircea, M., Uscatu, C., & Ghilic-Micu, B. (2016, 02 4). Analyzing Agile Development – from Waterfall Style to Scrumban. *Informatica Economică, 20*.

Sulaiman, N., MohdNaz'ri, M., & RasimahCheMohd, Y. (2016, 10 17). Influential Factors on the Awareness of Agile Software Development Methodology: A Systematic Literature Review. *Journal of Internet Computing and Services*, pp. 161-172.

Swaminathan, B., & Jain, K. (2012). Implementing the Lean concepts of Continuous Improvement and Flow on an Agile Software Development Project - An Industrial Case Study. *Agile India*, (pp. 10-19). Bengaluru.

Wallstrom, P., & Chroneer, D. (2016, 09 20). Exploring Waste and Value in a Lean Context. *International Journal of Business and Management, 11*(10), pp. 282-297.

Wang, X., Conboy, K., & Cawley, O. (2012, 01 31). "Leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *The Journal of Systems and Software*.

Vidyanand, C. (2007). Comparison of Software Quality Under Perpetual Licensing and Software as a Service. *Journal of Management Information Systems, 24*(2), pp. 141-165.

Womack, J., & Jones, D. (2003). *Lean Thinking* (Vol. 2). Free press.

Womack, J., Jones, D., & Roos, D. (1990). *The machine that changed the world.* USA: Macmillan.

Yin, R. K. (2009). *Case Study Research, Design and Methods* (Vol. 5). United States of America: Sage.

Appendix A: Table used to summarize the transcribed interviews

	Participants	Team				
	Identified, Yes/No	Description	Approach to minimize	Frequency, how often?	Results	Evaluation and improvement
Partially done work						
Extra features						
Hand-offs						
Task switching						
Relearning						
Delays						
Defects						
Most detrimental waste						
Extra intresting obs						

Appendix B: Pattern matching analysis summary – Improved waste elimination approaches

Aspect	Findings in literature	Current approach	Suggestions
Awareness	Understanding the concept waste (Poppendieck & Poppendieck, 2003; Al-Baik & Miller, 2014)	Low understanding of the concept waste	Increase the understanding: teaching sessions and material sent out to all the units employees.
	Classification model creates a mindset, reinforcing the habit of seeing waste (Poppendieck & Poppendieck, 2003; Al-Baik & Miller, 2014)	No classification or definition of waste	Introduce Poppendieck & Poppendieck's (2006) waste classification model
	Senior management and employee support (Al-Baik & Miller, 2014)	Senior management support	Gain employee support by making significant early improvements
Indication	Clear, reliable and automatically collectable measures (Staron, 2012)	Clear, reliable and automatically collectable measures	Continue using Jira for measures and indicators
	Provide foundation for analysis (Petersen & Wholin, 2010)	Shallow, one-dimensional that does not provide means for further analysis	More sophisticated measures and indicators that provides foundation for analysis.
	Visualized (Petersen, 2012)	Not visualized	Visualize indicators
	Partially done work in consideration to capacity (SPC) (Petersen & Wholin, 2010; Petersen, 2012)	Subset of Partially done work indicated with some regards to capacity	Indicate Partially done work with SPC and have discussions regarding capacity
	Flow (cumulative flow diagram) (Petersen & Wholin, 2010, 2011; Petersen, 2012)	No measures or indicators of flow	Introduce Cumulative flow diagrams in order to visualize flow in development processes
	Lead time (box-plots) (Petersen & Wholin, 2010; Petersen, 2012)	No measures or indicators of lead-time	Introduce lead-time as measure, visualized as box-plots

	VSM (Poppendieck & Poppendieck, 2006; Petersen, 2010, 2012)	-	Introduce VSM to Integration, BOD and BE
Analysis	5 Whys (root cause analysis) (Miller & Al-Baik, 2016)	Avoidance of problems. Easy solutions	Establish root cause analysis by 5 whys
	Indicators analyzed combined (Petersen, 2012)	Not analyzed combined	Analyze indicators combined in order to make improvements and not changes.
	Indicators analyzed with consideration to quality (Petersen & Wholin, 2010)	Comprehensive quality measures	Analyze indicators with consideration to established quality measures
	End to end perspective of value stream (Poppendieck & Cusumano, 2012; Petersen & Wohlin, 2010)	Generally micro perspective and technical aspects of the process	Analysis of waste should consider an end to end perspective of value streams
Elimination	Practices adapted to context (Poppendieck & Poppendieck, 2003; Al-Baik & Miller, 2014)	POC and Peer review	Continue using POC & Peer review when needed
	Polices and standards, self-determinded, continuously improved (Miller & Al-Baik, 2016; Kondo, 2000)	Polices and standards at varied degrees continuously improved	Policies and standard managed with BI as example to follow
	Identification and elimination iteratively (Al-Baik & Miller, 2014; Petersen & Wohlin, 2010)	Iterative approach to improvements	Continue using iterative approach to improvements
	Continuous improvements (Swaminathan & Jain, 2012; Miller & Al-Baik, 2016)	Loosely structured improvements and follow-ups	Systematic and well defined approach to improvements with follow-ups
	Reflective practices (Miller & Al-Baik, 2016)	Cookbook facilitates reflection on action.	Cookbook managed with BI as example to follow
	Double-loop learning (Miller & Al-Baik, 2016)	-	Introduce the concept Double loop learning

Appendix C: Interview guide

Structure

The following interview template is structured accordingly for each waste:

- Can you identify this waste in your team?
- How does this waste manifest itself in your team?
- How does this waste affect your team's software development?
- What is your approach to minimize this waste?
- What result does it lead to?
- How do you evaluate and improve your approach to this waste?

General questions

Have you read and understood the sent out material? Any questions regarding the material?

In general, what is the most problematic obstacle in your development process?

Partially done work

Uncoded documentation

Unsynchronized code

Untested code

Undocumented code

Undeployed code

Extra features

Features not asked for by the customer

Prioritize value generating features

Relearning

Capturing important knowledge generated

Limit relearning

Engage knowledge in the development process

Handoffs

Limit amount of handoffs

Problematic handoffs

Task switching

Single projects/tasks

Not plan above capacity

Developers responsible for their "own" defects

Delays

Delays/waiting

Dependencies