STS11010
Examensarbete 30 hp
Februari 2011

Achieving Lean Software Development

Implementation of Agile and Lean Practices in a Manufacturing-Oriented Organization

Thomas Norrmalm

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Achieving Lean Software Development: Implementation of Agile and Lean Practices in a Manufacturing-Oriented Organization

Thomas Norrmalm

The study reveals improvement areas in terms of lead time and quality in a traditional software development process of a large manufacturing-oriented organization, and identifies four obstacles to the application of a Lean software development framework in order to achieve such improvements. The data from interviews are matched to four predefined categories. These categories are evaluated using value stream mapping and a framework of seven common improvement areas in software development. A large project and task tracking system indicate that lead time is a real problem in the process. The most significant improvement area is wait time for change approval meetings. A second prominent improvement area is the large amount of approval handshakes. At least a few of these handshakes are always approved, thus adding unnecessary lead time to the process.

The four most imminent obstacles in adopting lean software development are identified through estimating the efficiency of two in-house derivations of Scrum and Kanban. The first obstacle is deep vertical but narrow horizontal expertise among developers. With some systems, there's only one developer who knows how to maintain the product. This makes it impossible to work as a team which is an imperative principle of lean. A second obstacle is how the teams are arranged organizationally. They have a functional setup over three departments and three managers, which to some extent create a silo mentality, rendering cooperation difficult. A third obstacle is how the teams are arranged geographically. Split over two locations, manufacturing and headquarters, they have different customers, objectives and a plain unfamiliarity with another that has reduced the will and opportunity to communicate and coordinate. A fourth obstacle is the inherent conflict between the prescriptive activities of ITIL, optimized for IT operational services, and the adaptability of agile methodologies, optimized for rapid change and empirical decisions. ITIL fulfills a sometimes uncalled for need to get all changes approved through several layers of management.

The study concludes that Lean software development is in conflict with many traditional values of a manufacturing organization. Although lean may be prevalent in other parts of the organization, this does not necessarily include the IT function. IT still seems to have hard time grasping the lean concepts of flow, waste and value.

Handledare: Tommy Hurtig
Ämnesgranskare: Roland Bol
Examinator: Elisabet Andrésdóttir
ISSN: 1650-8319, STS11010

Sammanfattning

Studien visar på förbättringspotential inom ledtid och kvalitet i en traditionell mjukvaruutvecklingsprocess hos en stor tillverkningsorganisation. Studien identifierar också fyra hinder för att applicera ett Lean software development-ramverk i syfte att åstadkomma sådana förbättringar. Data från intervjuer matchas mot fyra kategorier. Kategorierna är utvärderade med hjälp av värdeflödeskartor och ett ramverk med sju vanliga förbättringsområden i mjukvaruutveckling. Ett omfattande projekthanteringsverktyg indikerar att långa ledtider är ett reellt problem i processen. Det tydligaste förbättringsområdet är väntetid inför möten där förändringar ska godkännas. Ett andra tydligt förbättringsområde är det stora antalet godkännandeförfrågningar vid olika faser i processen. Åtminstone några av förfrågningarna godkänns alltid och förlänger således processens ledtid i onödan.

De fyra mest omedelbara hindren för en anpassning till Lean software development presenteras genom att bedöma effektiviteten i två egenutvecklade metoder baserade på Scrum och Kanban. Första hindret är en hos utvecklarna djup vertikal men smal horisontell expertis. I vissa system är det blott en utvecklare som har kunskapen att hantera systemet. Detta gör det omöjligt att arbeta i grupp, vilket är tvingande del av lean. Ett andra hinder är hur grupperna är arrangerade organisatoriskt. De har en funktionell uppdelning över tre avdelningar och tre chefer, vilket till viss del skapar ett silotänkande och gör samarbete svårare. Ett tredje hinder är hur grupperna fördelas geografiskt. Eftersom de är spridda över två platser, huvudkontoret och fabriken, och därmed har olika kunder och målsättningar, och dessutom en allmänt låg förtrogenhet med varandra, så är incitamentet för kommunikation och koordinering ännu lägre. Ett fjärde hinder är den inneboende konflikten mellan reglerade aktiviteter i ITIL, som är anpassade för IT-drift, och dynamiken i agil utveckling, som är optimerad för förändring och korta beslutsvägar. ITIL skapar ibland ett onödigt behov av att auktorisera förändringar genom flera nivåer av beslutsfattande.

Studiens slutsats är att Lean software development är i konflikt med många traditionella värderingar som man hittar i en tillverkningsorganisation. Trots att lean existerar i vissa delar av organisationen så behöver inte detta betyda att IT-organisationen tagit till sig av dess filosofi. Det tycks fortfarande vara svårt för IT att anamma lean-koncept som flow, waste och value.

Acknowledgements

The paper you are holding in your hand was written during a 20 week period at the ITT Water & Wastewater headquarters in Stockholm. A lot of people helped me before, during, and after the study was conducted, too many to mention them all. I would, however, like to thank a few people who had a particularly important role in making the study possible.

First I would like to thank Tommy Hurtig, my supervisor at Water & Wastewater, for giving me the opportunity to do this study, and supplying me with every resource I could possibly need. Secondly I would like to thank my reviewer, Roland Bol, for providing guidance and valuable help with how to approach the scientific area of lean and agile. As a third and final thank you I would like to mention Britta Schreiber, teamleader at Water & Wastewater, who was an invaluable source in describing the ways of work at the company.

Thank you!

Stockholm, January 2011

Thomas Norrmalm

Table of Contents

Glossary

Please use this list for quick reference.

COBOL	*Common Business-Oriented Language* – a programming language seen in corporate data centers and mainframes
Department	An organizational term for entities directly under *functions*
Dot-net	*.NET Framework* – a software framework developed by Microsoft for Windows operating systems
Function	An organizational term for the entities directly under *W&WW value center*
IDMS	*Integrated Database Management System* – a database management system used to administrate the mainframe
IT operations	The group responsible for the day-to-day monitoring and management of IT Services and IT Infrastructure
ITT	*ITT Corporation* – Conglomerate acting in global markets including water and fluids management, defense and security, and motion and flow control. Is the parent company of W&WW
Methodology	A framework used to structure, plan, and control the process of software development
PULS2	Project management template used by all functions at W&WW except R&D
RFC	*Request For Change* – A written, formal description of a wanted change in a information system
SDP	*Software Development Process* – structure imposed on the development of a software product. Describes approaches to a variety of tasks or activities that take place during the process
SOA	*Service-Oriented Architecture* – a set of design principles used during the phases of systems development and integration
Team	A team comprises a group of people belonging to a *department*
UAT	*User-Acceptance Test* – Process to obtain confirmation that a system meets mutually agreed-upon requirements
W&WW	*Water & Wastewater* – global provider of water handling and treatment solutions for municipal and industrial customers in more than 140 countries

1. Introduction

Think about it. A cell phone, computer, laptop – perhaps even a tablet – you have one. Are they at work, at home, by the TV, by the kitchen table or just everywhere? IT appliances are becoming an inseparable part of our daily lives. Still, for most of us, these appliances are just a "thing", a *black box* that enables a certain action in a certain context. To buy a train ticket, for example. We surf the web, enter a few text strings and voilà, a ticket is generated on the screen. At least most of the time. When this black box fails us, however – may it be due to loss of connection, a bug in the software, or incompatibilities with the browser – we become distressed. But because we know that life without IT systems is no longer possible, we calm down just as quick and restart the system. Suddenly it works again. The *software* works again.

This paper, in essence, is about how software can be made better. How do you make software better? You must improve the *process of making it*, of course. Over the years, many "best ways" of creating software have emerged and disappeared. The latest one to emerge is *agile*. Agile, in its general definition, is often described as

> *the ability to change the body's position efficiently, requiring the integration of isolated movement skills using a combination of balance, coordination, speed, reflexes, strength, endurance and stamina*

Does it sound promising as a way of creating software? Hard to tell, depends on what you compare it to. The traditional approach to software development is based on *manufacturing* principles. Manufacturing is defined as the act of

> *creating or producing in a mechanical way, or the transformation of raw materials into finished products*

How can these approaches be even remotely related, and why in software development? How do they *perform*, in comparison? That is the topic of this paper. One must, however, put these methodologies in context of history to fully understand their implications.

In 2008, the value of the global software industry was US$ 303.8 billion (Software: Global Industry Guide 2009). In 2013 the global software market is predicted to value US$ 457 billion, an increase of 50.5 percent. Although still growing, the software industry is already considered the third-largest U.S. manufacturing business, after automobiles and electronics. With such remarkable growth, and over 40 years[1] of software development in the making, you would expect a high degree of maturity in the production of software. This, however, is not the case. Software development projects are still closely tied to properties such as *poor quality*, *past deadline*, *over budget*, *buggy* – or simply *failure*. Some say a third of the projects are failures, some even two-thirds (Standish Chaos report 2003); in either case, the numbers are extraordinary. Imagine car manufacturing where only two thirds of the cars coming out of the factory are functional. How can this difference in success rate be explained?

[1] I consider the NATO sponsored conferences on software engineering in 1968 and 1969 to be the official start software development as it is known today.

A part of the explanation is found in the old manufacturing principles of Fordism and Taylorism (i.e. *the scientific method*) that still dominate the development of industrial products and software. Generic software development processes such as the *Software Engineering Institute's Capability Maturity Model* and *ISO 9000/9001* were conceived to standardize the software development process, exactly like they had standardized the manufacturing of traditional products. Discipline and professionalism were described as silver bullets. Unfortunately, the continuation of the *software crisis* into the 1980s revealed that the quest for perfect software still was not over (Brooks 1986).

The next approach to show promise was the Japanese principles of quality. With the addition of lean principles, the traditional manufacturing model of development was to be perfected, or at least that was the plan (Mah 2008). Reality, however, proved different. Rather than improving software development, today's projects are still performing as dire as they did in the 80s, with late deliveries and unfulfilled customer requirements still as common as ever (Mah 2008). Most researchers argue that there is no single approach that could resolve all complexities in software development (Brooks 1995).

In recent years, lean and agile principles have gained popularity in addressing problems such as long development life cycles and rapidly changing customer requirements. Poppendieck et al (2003, 2005 and 2009) has taken lean and agile thinking to propose a new software development methodology, *lean software development*, which combine core principles of delivering *customer value*, *minimize waste* and *maximize flow*. Properly implemented, lean software development promises increased productivity and reduced lead time (Bjørnvig et al 2010). The methodology proposes improvement particularly for medium to large companies with a history of lean in manufacturing, where the philosophies already imbue some parts of the organization (Poppendieck et al 2009). Although quantitative research on agile methodologies versus traditional methodologies is rather scarce, Mah (2008) and Ambler (2007) indicate that some impressive results may be achieved. Mah (2008) compares the performance of 23 agile projects against an industry average of 7,500 completed software projects in the Quantitative Software Management Associates database. He concludes that about 80 percent of the agile projects were completed quicker than similarly sized traditional projects. In terms of quality, traditional manufacturing type projects reported a higher defect rate when using large teams, sometimes as much as four times higher than the agile average. Mature agile project teams showed a defect rate that remains proportional to the project's increase in size (Mah 2008).

ITT Water & Wastewater (W&WW) has a long tradition in manufacturing and industrial applications. With 5800 employees, it is the world's largest supplier of pumps and systems for the transportation, treatment and control of water (W&WW official site 2010). The company is a part of the ITT Corporation conglomerate, a globally diversified manufacturing company with revenues of $11.7 billion in 2008. ITT participates in several other global markets, including defense and security, and motion and flow control (ITT official site 2010). W&WW consider software development important in keeping the in-house software competitive. The IT department is responsible for developing tools related to calculation, configuration, selection and design of complicated watering solutions that ships to resellers on a global scale. The software solutions aim to greatly simplify and automate work for sales, manufacturing and engineers on-site. (Teamleader A, 2010)

The IT department, with a staff of approximately 150 employees, is based in Sundbyberg and Emmaboda in Sweden; the majority of software development also takes place in these two locations (Teamleader A, 2010). Today's development process is a traditional sequential waterfall process, based on processes from the manufacturing side of the company. Development progress is made by sequentially entering the phases of concept, initiation, analysis, design, construction, testing and maintenance. Lead times, however, are becoming increasingly long, and there is a prominent problem of projects being cancelled half-way through (Teamleader B, 2010). In recent years the manufacturing part of the company has begun working towards lean principles of management. This, however, has not been implemented in the software development process. This paper aims to shed light on the downsides and benefits of implementing lean software development practices in such a context.

1.1 Purpose

The purpose of this study is to contribute to the understanding of how lean and agile principles can be combined to reduce lead time in the software development division of a large manufacturing-oriented organization.

1.2 Aim

The aim of this study is to answer the following research questions:

- How is the software development at ITT W&WW organized?
- To what extent can a Lean software development framework improve the software development process in terms of minimizing waste?
- What are the most imminent obstacles in implementing such a framework?

The analysis is focused on the earlier parts of the software development process, mainly the change management process. The release and maintenance process will not be covered in any detail. Focus is on developer teams rather than architectural teams. With that said, knowledge had to be gained in how interaction between all stakeholders in the process work, thus requiring a general idea of how the other teams work as well.

1.3 Methodology

To successfully produce software is a *complex process*. A complex process has a large set of variables affecting its performance. What this thesis aim to do is to deliver a starting point to improve on those variables; a qualitative evaluation how the organization performs and what agile and lean principles can bring to reduce lead-time and improve quality over time.

The way of measuring performance in this paper is based on the Software Benchmarking Organization's (2010) definition. The Software Benchmarking Organization offers a set of best practice key performance indicators (KPIs) to measure performance in software development projects. They define software development performance as

> *the efficiency of the development process in terms of value-adding core activities, support activities, prevention activities (reviews, inspections) and appraisal/rework activities (testing, defect removal) (Benchmarking Organization, 2010)*

W&WW has no implementation of KPIs, and thus no quantitative measure to estimate the performance of the development organization (Teamleader B 2010). Therefore few numbers are available that indicate the efficiency of the current SDP. The aim of my findings is consequently to suggest a *relative* improvement over a previously unknown state.

1.3.1 Lean and Agile Principles

Central to the thesis is lean and agile principles (see 2.3 for a more comprehensive discussion on frameworks, concepts and definitions). By using the waste-value concept and related tools an organization can continuously measure and improve its processes. The main analysis tool of this thesis is the *value map stream,* which maps the value-adding activities of the SDP, and the *seven wastes of lean software development*, a set of waste activities that are commonly seen in development (Poppendieck et al 2003). The lean value map is used to give an overall view of the process, and the seven wastes serves as a guide to focus the effort of finding waste. It has been shown, as is also discussed in the introduction, that agile approaches seldom decrease performance over a traditional manufacturing-inspired approach. Therefore, by using light-weight agile methodologies as a *good practice* comparison, arguments can be presented how waste in the process can be reduced.

A number of sources contributed to my interpretation of agile and lean, and their unification in the approach called lean software development. Mary and Tom Poppendieck (2003, 2007 and 2010) are the founders of the lean software movement. Their work merge lean manufacturing, lean IT and agile, and they present their insights from both a managerial and developing viewpoint. Mixing them with the books by Beck (2004), Bjørnvig et al (2010), Cohen (2010) and numerous Internet resources gave me an in-depth understanding of how agile projects can add value in practically any organization. It also provided, perhaps more importantly, scenarios where agile performs less well.

The Internet is, as always when studying an IT related subject, a vivid resource of up-to-date reports, literature and perceptions. Lean software development is no exception to this rule. The research area is still very active with many blogs, reports, white papers and formal and informal communities that discuss topics on a day-to-day basis. The works of Mikael Lundgren (2010) and Henrik Kniberg and Mattias Skarin (2010) are particularly valuable in my understanding of agile methodologies Kanban and Scrum in both practice and theory.

Many of the front figures in the community do agile coaching for a living and as such, they may have a tendency to favor the techniques they teach. These subjective opinions were neutralized to the extent possible by cross-checking numbers, and always keeping a critical standpoint in regard to unverifiable data. It is, however, very hard to find anyone within software development who advocates the traditional waterfall approach. The general standpoint is therefore that a properly implemented agile approach *is* more efficient than the traditional approach under most circumstances.

The author's objectivity towards W&WW can be questioned; I spent two months working in one of the projects taking place in the IT department during 2010. I do argue, however, that the company has a welcoming and open approach and that my previous experiences as an employee not directly interfere with the purpose of this thesis. On the contrary, I think the background I had when starting this thesis was very

valuable. The teams were somewhat familiar with me which, in terms of trust, gave me a head start.

1.3.2 Timeline and Course of Action

The data of this study is collected through four methods: a *literature review* (see discussion above), *interviews* and *observation*. The majority of data-collection took place at the W&WW headquarters in Sundbyberg, Stockholm, over a time-span of approximately 20 weeks. The thesis had a pre-established project plan, most of which was kept in phase with (see table 1).

Table 1. Course of action, from initial stages to analysis. (Source: Author)

#	**Purpose**	**Activity**	**Subject**
1	Literature study. Overview of the accumulated knowledge in lean and agile	Reading of books, reports, white papers and blogs	Myself
2	Overview of the ITT conglomerate, down to the teams on different branches of the IT organization	Structure thoughts and experiences from my previous work at W&WW. Write down realizations and create categorization	Myself
3	Gain in-depth understanding of what W&WW software development aims to achieve. Basic understanding of the different teams and their methodologies	Overview interview, semi-structured interviews. Adjust categorizations after new realizations and sort data accordingly	Teamleaders, developers
4	Gain understanding of the Sundbyberg developer team and their way of work	Semi-structured interviews, sort data in categories	Sundbyberg developers and teamleaders
5	Gain understanding of the Emmaboda developer team and their way of work	Semi-structured interviews, sort data in categories	Emmaboda developers and teamleaders
6	Establish process efficiency and find bottlenecks	Retrieve information from *processes* category. Map value stream and analysis	Continuous meetings with one teamleader
7	Analyze the organization in relation to the seven wastes framework	Retrieve information from *team* and *self-reflection* categories. Analysis of results	Myself and feedback from teamleader

Although literature reviews (table 1, step 1) were the initial step, most data collected in this paper emerged as an iterative cycle between theoretical conceptions and empirical data. This means that during empirical data collection, I continuously reviewed my data against the literature and revisited sources when in need of clarification.

During step 2, the subjects to interview were selected. The aim of my selection was to capture the views and opinions of managers as well as developers. Once selection was done, initial contact was taken through mail, direct communication or over the phone. All of the interviews were semi-structured; I sketched a few topics that were relevant to what I wanted to know about a specific activity in the software development process, and the topics were given beforehand to the interviewee. Topics were structured in accordance to the role of the subject (see Appendix V for detailed topics):

- Teamleaders were mainly asked about their team's purpose and workflow, and how they cooperated and coordinated with other teams
- Developers were asked how they experience their current work situation, and how they worked practically with tools and computers
- Managers were asked how they staffed projects, and how priorities between different tasks were done

The collected data were categorized by type: *self-reflection*, *team*, *development process* and *overall process*. These categories were created to simplify the interview process, and align data with the lean software development framework. The self-reflection category included data on how the interviewees reflect on their own process and lean. Team category included data on teams' purpose, setup and external collaboration. The development process category and overall process category included data on the specific build and test process and the entire process, respectively. When referred to in the paper, all interviewees are anonymous and gender neutralized. Source references are made with the organizational role of the interviewee and a letter.

After the interviews were done, the next phase took place (see table 1, step 6). At this stage my understanding of workflows and processes was sufficient. The next step was to map the actual flow using the value map. This was done in cooperation with one of the teamleaders. She helped me estimate activities, wait-time and work-time in the entire SDP. As the last phase, step 7, continuous meetings were held with the team leader of the Sundbyberg developer team in order to map, present and discuss the seven wastes. The whole approach can be visualized as in figure 1.

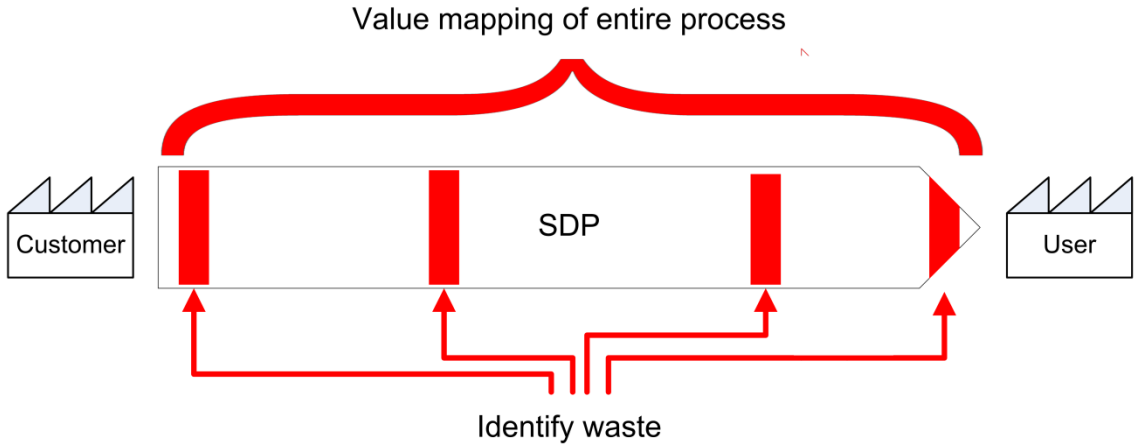


Figure 1. Approach to analyzing the SDP. First a value map overview of the whole process, then focus on particular bottlenecks and wastes in the flow. (Source: Author)

1.4 Disposition

The paper is divided into five chapters. The introduction chapter discusses software development over time, and how agile and lean principles are growing in popularity, promising improvement over traditional software development processes that often are described as inherited from manufacturing. The methodology of the paper, including value stream map and seven wastes, is presented.

Chapter two aims to discuss the software development process, i.e. investigate the kind of problems encountered in this domain. Next are the concepts of *process complexity* and *prescriptive-adaptive methodologies*, which are useful to understand the comparison between the traditional waterfall approach and the agile initiative. Scrum and Kanban, two agile methodologies, are presented in 2.2. In the last part of chapter two, lean and agile principles are combined into the concept of *lean software development*, and based on that framework the lean value map and seven wastes are discussed.

Chapter three is a case study of software development and its context at W&WW. Part one is an overview of the company, the business divisions and, lastly, the W&WW value center. Next part is a look into the teams of the SDP, with focus on their composition, methodology and internal communication. Next section gives insight into the frameworks directly affecting the SDP, i.e. ITIL best practice and PULS2 project management. Lastly, the whole SDP is presented, starting with customers initiating a software change and ending with the user actually being affected by the new update or system.

Chapter four is the analysis part, starting out with a value stream map analysis, where bottlenecks are identified. With the value map in mind, the next section focuses on the seven wastes of software development. The third part analyses the current agile implementations and how they perform. Fourth is a section with proposed actions that, based on lean assumptions, would improve the performance of the SDP.

Chapter five offers the conclusions of the thesis, including suggestions for further studies.

2. Theoretical Framework

In order to effectively discuss lean and agile concepts, one must carefully define their meaning and context of application. The purpose of the following chapter is – in chronological order – first to explain the generic software development process and some basic attributes of it, secondly to compare the traditional manufacturing-inspired waterfall approach with the light-weight agile initiative, third is a study of agile methodologies Scrum and Kanban. With the previous in mind, 2.3 merges lean core principles and agile principles into lean software development. At this point, the reader should have a basic understanding of lean, agile, Scrum and Kanban that is similar to figure 2.

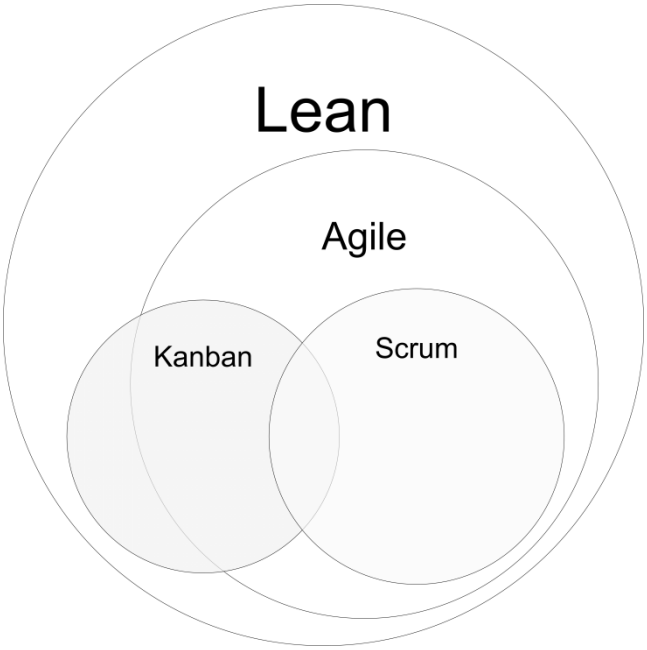


Figure 2. Relational diagram including lean, agile, Kanban and Scrum. Lean provides the basic framework for agile principles, and Kanban and Scrum are methodologies that apply the principles of lean software development on the SDP. (Source: Author)

2.1 Aspects of a Software Development Process (SDP)

This section is a brief overview of the different attributes of a software development process (SDP), including terminology and definitions. A SDP is, in its most basic form, a set of activities that span requirements, specification, architecture, design, implementation, testing, deployment and maintenance (Bjørnvig 2010). It always starts with some kind of idea. The idea can come from *internal channels*, i.e. from customers within the organization, or *external channels*, i.e. from customers external to the organization (Bjørnvig 2010). This thesis primarily focuses on SDPs with input from internal channels, where customers and users are sales, marketing, R&D or other in-house functions. It is important to note the distinction between *customers* and *users*. The customer is defined as the person directly engaged in the development effort, i.e. the one that pays for the work done. The user, on the other hand, is the person who uses the system. Another stakeholder in the process is the *developer*. The role of a developer in this paper follows the definition of Lundgren (2010). A developer is anyone who contributes to the functionality and value of the final product. This includes – but is not

limited to – programmers, test specialists, database specialists, and UI designers. At the end of a complete SDP, a *product* is delivered to the customer. The product can be an entirely new application, or an update to an already existing application (Bjørnvig et al 2010). Software bugs are referred to as *defects*. As such, one quality indicator of a product is the number of defects in the final product (Poppendieck 2003).

The way that the above mentioned activities, roles, and procedures are set up is referred to as the *methodology* of the SDP. Not all parts of a methodology are formalized; there are routines and procedures that are tacit knowledge (Beck 2004). Where a methodology is not at all formalized, I will refer to it as an *ad-hoc methodology*, meaning it is all based on direct, empirical decisions rather than on any particular framework.

2.1.1 Process Complexity

Software development organizations must adapt its approach to the elements of the system being built and the stakeholders building it. These elements are the *technology*, the *requirements*, and the *people* involved in the creative process (Schwaber 2010). The interactions between these elements are the basis of the *complexity* concept. Schwaber (2010) argues that all development projects can be described using two dimensions: simple to complicated and fully predictable to chaotic (see figure 3).

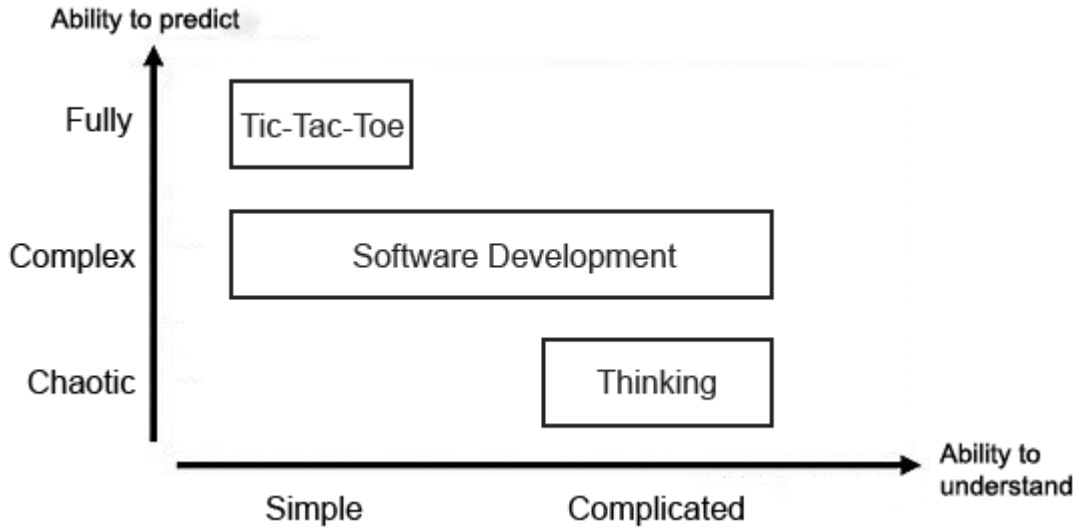


Figure 3. Systems are predictable, complex or chaotic. Regardless of which, they are also simple or complicated to understand. Tic-tac-toe is simple, and human thinking is chaotic. Software development is always complex, but can still be simple or complicated as a system. (Source: Appelo 2008)

The definition of a simple system is that it's *easily understandable*. It has a limited number of components with a limited amount of interaction, and as such it can be fully understood. A complicated, fully predictable system on the other hand, is not simple, but still *understandable* – the number of components and interactions are larger but still limited. Both simple and complicated problems can be *predictable*, *complex* or *chaotic* (Appelo 2008). A complex system is not fully predictable, but still predictable to a certain degree. A chaotic system is never predictable. (Schwaber 2010)

The difference between predictable and complex systems is the approach to understanding them. Simple and complicated systems can be understood by analysis of

parts and interactions. Complex system can only be understood by studying how the *whole system operates*, never by identifying all parts of it. Software projects, whether simple or complicated, are always complex (Appelo 2008).

2.1.2 Prescriptive and Adaptive Methodologies

The methodology, as defined in 2.1, is essentially a framework which describes the logical process of constructing a system. It has phases of defining, building and implementation. When the complexity of the system's problem domain is established, an appropriate methodology can be chosen (Scrum methodology 2009). A system can be *theoretical* or *empirical*. For a system to be classified as a theoretical system, it must be derived from *first principles*, i.e. mathematical axioms. Only if it conforms to these constructs can it be considered a theoretical process. Empirical modeling on the other hand, relies on observed inputs and outputs without depending on any laws during the construction process. It strictly depends on experimentally obtained information, i.e. empirical evidence (Scrum methodology 2009). Theoretical modeling only involves the estimation of the unknown parameters in a system. This means that fewer measurements are necessary for the system to be modeled correctly. This type of mathematically valid methodology is called *formal*. Outside of academy and research, it is not very common. (Knight et al 1997)

A formal methodology usually has a higher *prescription* level then empirical methodologies. The prescriptions are essentially *constraints*, or rules, that must be followed when applying a methodology to a process. Fewer constraints imply a more adaptive (empirical) methodology, and vice versa (see figure 4).

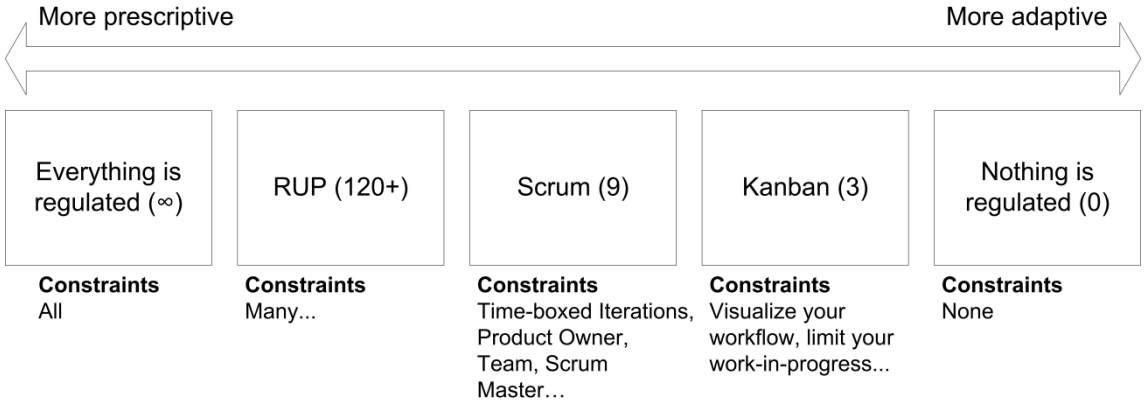


Figure 4. A comparison between methodologies on a prescriptive to adaptive scale. Three constraints (as in Kanban) mean you are free to adapt the methodology as you prefer – as long as you comply with the three constraints. Scrum has nine constraints, meaning it is more prescriptive than Kanban, but still highly adaptive. The rational unified process (RUP) methodology is included for comparison. RUP is a highly prescriptive methodology with over 120 constraints in the process. (Source: Author, based on Kniberg & Skarin 2010)

Generally, the more theoretical (and thus predictive) a system is, the more suitable is a prescriptive (plan-driven) methodology. Adaptive methodologies focus on adapting quickly to changing requirements. An adaptive team never goes into detail exactly what tasks are being done next week, but only which features are planned for next month. As a consequence of this, adaptive teams will have difficulty describing exactly what will happen outside of a given time horizon. (Appelo 2008)

Prescriptive methodologies, in contrast, focus on planning the future in detail. A team using a prescriptive methodology knows to a high degree what features and tasks are planned for the entire length of the development process. Instead of being adaptive, prescriptive teams have difficulty changing direction with changing requirements. The project plan is optimized for the original target and changing direction can require completed work to be started over, and thus cause much waste in the process. Prescriptive teams will often institute a change control board to ensure that only the most valuable changes are considered. (Boehm et al 2004)

Often the methodology of work is justified by the supposed ambition to maximize or minimize revenue or similar business variables. Companies are seldom scientifically motivated to choose development methodology. Fear of change and tradition – *the way we have always done it* – are also common reasons why an organization works with a specific methodology. (Knight et al 1997)

2.1.3 The Traditional Waterfall Approach

Most SDPs have historically used some derivation of the prescriptive waterfall methodology, mainly because software development originated in the manufacturing and construction industry of the 1960s. At that time, no formal development methodology existed (simply because software development was unheard of), so the solution was to apply the processes from the manufacturing side of the company on software development. (McConnell 2004)

In a manufacturing plant, a workflow is laid out to turn raw material into a finished product. When the workflow is established and preparations are done, managers are hired. The managers bring in resources (preferably automated) to staff the various points in the workflow. Then the job of the manager and her supervisors is then to instruct the resources how to do a pre-defined task in the process. If the resources handle their work correctly, the flow is constant and the products are manufactured as planned. (Schwaber

In such environments, with highly defined processes, *big design up front* (BDUF) and *big requirements up front* (BRUF) suits the business well when building or buying machines. The business need is usually well-defined, with industrial machines only having one or a few specified purposes. How the machines should perform is relatively easy to calculate (McConnell 1996). BDUF and BRUF make sure that the design is completed and perfected before the actual construction starts. As such, it helps prevent expensive changes later on in the process. This is a successful approach with industrial applications. McConnell (1996) estimates that a defect left undetected until construction or maintenance would cost fifty to two-hundred times as much to fix

The concept of calling this manufacturing-inspired methodology a *waterfall*, with BDUF being a large part of it, was coined in 1970 by Royce (1970). In the book, a sequential process similar to a waterfall is presented (see figure 5).

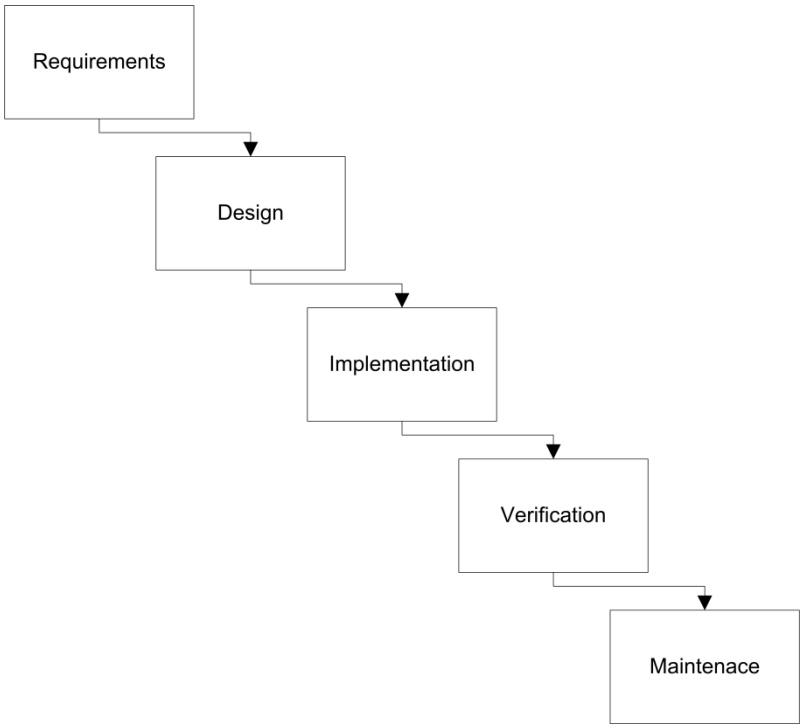


Figure 5. The waterfall process is an irreversible sequence with isolated stages ranging requirements, design, implementation, verification and maintenance. The model is sometimes presented with feedback loops as well, i.e. verification has a channel back to implementation in order to correct the errors found during testing. (Source: Royce 1970).

He describes the model as a flawed, non-working process. Before following the flow down, the current phase has to be fully complete. An incomplete phase creates defects in the final product. Given that it's impossible to revert back to the previous phase, a lot of work and resource planning then has to be done up front (Royce 1970). Because the waterfall approach acts mostly as an example of a failed development process, questions have been raised how much it is actually used in real-life projects. Weisert (2002) argues that the waterfall approach is practically non-existent in modern software development because it never existed (Weisert 2002). His arguments conclude that a waterfall process, where it is impossible to go upstream or reverse, simply cannot exist in real-world project management. Instead he argues that the most common use of the waterfall process is to scare companies into using new methodologies that require training and money (Weisert 2002). This critique may or may not be valid, but it is still important to comprehend the waterfall process as made up of, at least to some degree, isolated steps where BDUF and BRUF plays an important role. Although most organizations not apply the exact waterfall methodology described here, many development efforts still bear great resemblance to the step-by-step approach that it prescribes (Mah 2008).

2.1.4 The Light-Weight Agile Initiative

The Agile initiative was started as a reaction to the traditional, manufacturing-inspired way of creating software. It is, however, not a single methodology but rather a collection of methodologies. Some parts of it draw from lean, and some from the very earliest years of software engineering (Weinberg 2003). What is shared among the methodologies is the focus on iterative and incremental development, where

"requirements and solutions progress through collaboration between self-organizing, cross-functional teams" (Bjornvig et al. 2010). Agile methodologies are considered light-weight and adaptive (in terms of prescribed activities) in comparison to traditional methodologies, which often are characterized as over-prescriptive and over-managed (Bjornvig et al. 2010). The term *agile* was coined during a gathering between seventeen veteran software developers in 2001. Their manifesto, in its entirety, was written as

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> *Individuals and interactions over processes and tools*
> *Working software over comprehensive documentation*
> *Customer collaboration over contract negotiation*
> *Responding to change over following a plan*
>
> *That is, while there is value in the items on the right, we value the items on the left more (Agile Alliance official site 2010)*

The manifesto serves as the core of the agile movement. Today the Agile Alliance – a non-profit organization founded by developers from the agile gathering – promotes software development according to the manifesto's principles (Agile Alliance official site 2010).

Twelve Principles

One of the developers behind the agile initiative, Kent Beck (2001), translated the manifesto into the *twelve principles of agile*, a set of principles which specifies the concrete meaning behind the manifesto. The principles are (Manifesto for Agile Software Development 2001):

- Early and continuous delivery of valuable software
- Welcome changing requirements, even late in development
- Deliver working software frequently, with preference to the shorter timescale
- Business people and developers must work together daily
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done
- The most effective method of communication is face-to-face conversation
- Working software is the primary measure of progress
- Sustainable development: developers, sponsors and users should be able to maintain a constant pace
- Continuous attention to technical excellence and good design
- Simplicity – the art of maximizing the amount of work not done – is essential
- Great architecture, requirements and designs emerge from self-organizing teams
- Reflect on how to become more effective, then adjust behavior accordingly

2.2 Agile Software Development Methodologies

Agile includes a wide array of methodologies. Two of the most commonly used are Scrum and Kanban, both said to honor the agile principles we saw in 2.1.4. They were, however, devised earlier than the agile initiative itself. The following section discuss and define some of their basic characteristics.

2.2.1 Scrum

Scrum is a time-boxed, iterative and incremental approach that aims to deliver on time and with the customer demands in focus. Each iteration is time-boxed in, typically, a two to four week long period (the length is derived by the team) where, at the end, the team must always present a potentially shippable product increment. This period is called a *sprint* (Cohen 2010).

A key principle in Scrum is how the customer requirements are handled. The basic assumption is that customers will (and should want to) change their minds about what they want and need during the length of development effort. Such unpredicted challenges cannot be easily addressed in a traditional predictive or planned manner. Instead, Scrum adopts an adaptive empirical approach, accepting that the problem cannot be fully understood or defined, focusing instead on maximizing the team's ability to deliver quickly and respond to emerging requirements (Cohen 2010). As a framework (see figure 6), it is constrained by a number of roles and activities. The roles of the team are:

- *The Scrum master*, who maintains the processes (similar to the role of a traditional project manager). The Scrum master is not leading the team (as the team is self-organizing) but acts as a wall between the team and any external disturbance
- *The product owner*, who represents the stakeholders and the business
- *The team*, a cross-functional group of about seven people. The team includes all the developers who do the actual analysis, design, implementation and testing

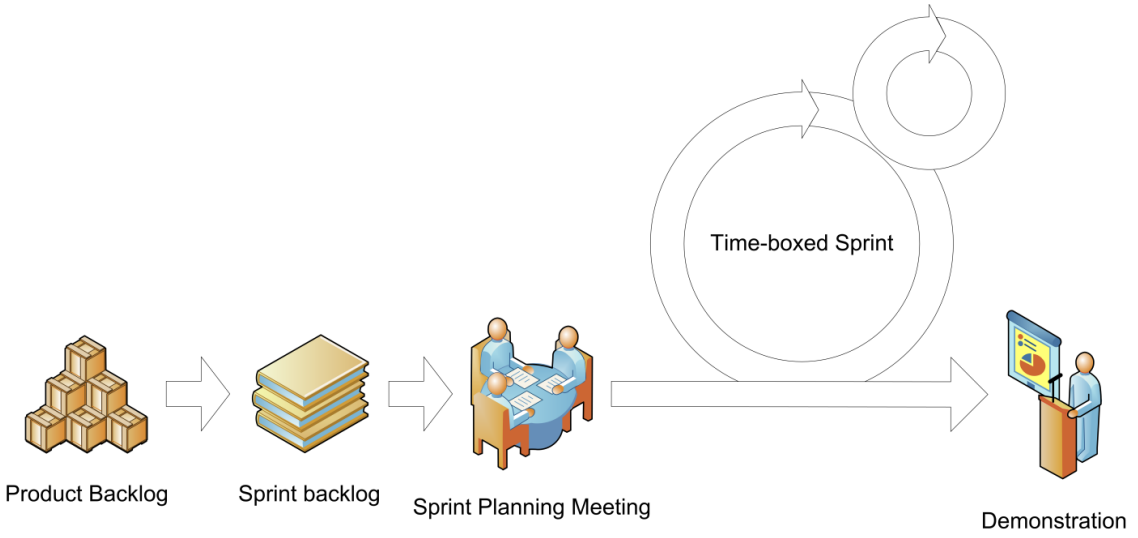


Figure 6. An overview of the Scrum methodology, including product backlog, sprint backlog, sprint planning meeting, time-boxed sprint and final demonstration. (Source: Author, modified from Cohen, 2010)

The set of features that go into a sprint is selected from the *product backlog*, which is a prioritized set (based on business value) of requirements to be developed. The *product owner* selects the tasks and presents them at the *sprint planning meeting* at the start of each sprint. During this meeting, the product owner informs the team of the items in the product backlog that she wants completed. The team then estimates how much of this they can commit to during the next sprint. (Schwaber 2004)

During a sprint, the requirements are frozen, i.e. no one is allowed to make any change to the sprint backlog. Each morning a short standup meeting is held, where the team can synchronize and planning is updated. In 15 minutes, the following questions should be answered by all team members (Cohen 2010):

- What have you done since yesterday?
- What are you planning to do now?
- What obstacles stops you from moving on?

If the proposed feature is not complete at the end of the sprint, it must be cycled back to the product backlog (Lundgren 2010). After a sprint is completed, the team demonstrates how to use the software. Time is put aside for a retrospective, where the team reflects on the work that has been done during the last sprint. These reflections are then used to adapt and improve the work flow until the next sprint (Dubakov et al 2008).

Scrum can be implemented using a wide range of tools. Many companies use software tools, such as Microsoft Excel, to build and maintain the sprint backlog (Dubakov et al 2008). There are also other open-source and proprietary software packages dedicated to management of products under the Scrum process. Other organizations implement Scrum without the use of any software tools, and maintain their artifacts on paper, whiteboards, and sticky notes. In aiding comprehension and teamwork, the whiteboard is generally the preferred choice (Dubakov et al 2008).

2.2.2 Kanban

The Kanban board, with its origin in lean manufacturing, has seen a recent popularity growth in software development. The name itself literally means *signboard* or *billboard* (Kniberg et al 2010), and that describes its main purpose rather well. In software development, and particularly agile software development, the visualization of projects on boards is a practice to structure task handling. In Scrum tasks are placed on boards for time-boxed sprints. Kanban is a different way to approach this. Instead of being time-boxed, tasks are *pulled* at any time, only limited by a work-in-progress (WIP) limit that restricts the allowed number of tasks in every workflow state (see figure 7).

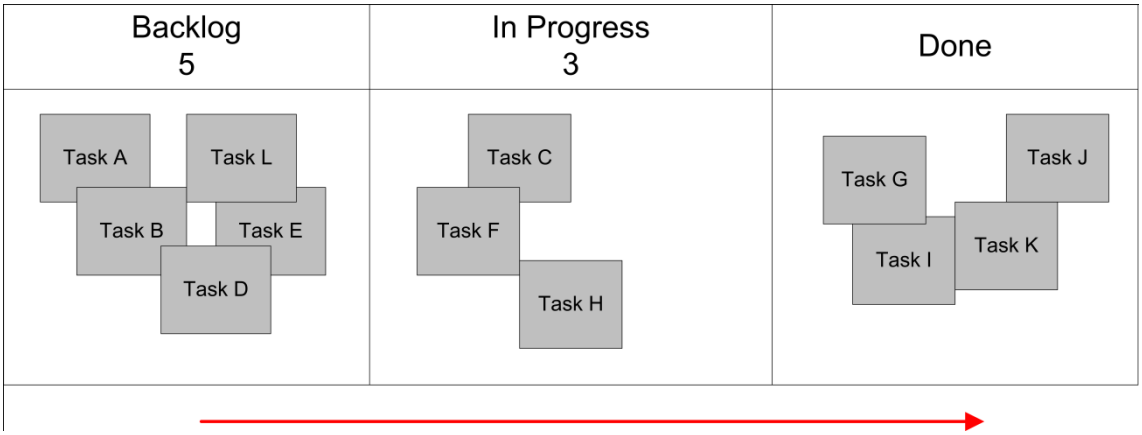


Figure 7. A Kanban board, with workflow states "Backlog", "In Progress" and "Done". Numbers under each workflow state title illustrate the work-in-progress limit of that state. The arrow indicate a flow from left to right, meaning that each task is moved , (Source: Kniberg & Skarin 2010).

The purpose of the pull-schedule is to maximize flow and reveal wait states. Prescriptions in Kanban include only three activities (Kniberg et al 2010):

- *Visualize the workflow*, split the work into items, write each item on a card and put on the board. Use named columns to illustrate where each item is in the workflow
- *Limit work-in-progress (WIP)*, assign explicit limits to how many items may be in progress at each workflow state
- *Measure the lead time* (average time for the tasks to be completed), with the purpose of optimizing the process. Lead time should be made as small and predictable as possible

Because WIP is limited in a Kanban system, any item that becomes stuck in a workflow state will quickly choke the system. If enough tasks become stuck the whole process grinds to a halt. This usually has the effect of focusing the whole developer team on solving the problematic task and restore flow. (Kniberg et al 2010)

Risks and Criticism

Schwaber (2010) argues that there is a substantial risk of Kanban being used when an organization cannot (or dare not) adopt Scrum. Many of Scrum most difficult aspects – such that may require organizational changes – are avoided. He claims that

> *...managers are still in charge of telling people what to do. People can be interrupted at any time. People are still working in functional silos, preserving the jobs of functional managers (Schwaber 2010)*

Lundgren (2010) has noted a similar tendency in his work; Kanban boards often have workflow states such as design, code, test and ship, thus implying a sequence of events where functional teams will be doing different actions to the code as it flows through. He remarks

> *Congratulations, you've just re-invented waterfall development in your own organization, but under a new and catchy name (Lundgren 2010)*

Lundgren (2010) brings up another risk with Kanban: *variability in task size*. A consequence of software development is the fact that tasks will not be uniform in size, and that they may change in size during their lifetime. Lundgren (2010) argue that for a Kanban to work properly, all tasks need to be of a defined size and stay that way. There's no point in having a WIP limit of three tasks at a certain stage, if the first task will take two hours and the other two three days each, Lundgren (2010) argues. Donald Reinertsen argues that much waste in product development actually comes from trying to minimize the variability of development tasks, instead of accepting and managing it. Kanban used this way may put focus back on trying to remove variability (Reinertsen 2007).

2.3 Applying Lean to Agile Software Development

The previous sections go into detail on agile, Scrum and Kanban. In the following part, lean is discussed. The application of lean on agile software development, popularly called *lean software development*, is a useful concept; mainly because it offers the opportunity to use lean concepts such as waste and value. Lean software development is essentially a translation of lean manufacturing and lean IT principles to the software

development domain (Bjørnvig et al 2010). Many principles are shared among agile and lean, and recent years has seen lean software development gain a solid reputation within the agile community (Poppendieck et al 2009; Bjørnvig et al 2010). The purpose of the following section is to setup a framework for understanding lean software development and its synergies with agile, starting out with the core philosophy of lean manufacturing.

2.3.1 Lean Philosophy

Lean is a process management philosophy derived from the Toyota Production System (TPS). TPS evolved from 1940 onward, under the direction of Taiichi Ohno. Ohno observed a need for Toyota to improve the production system in order stay competitive in the crowded Japanese market. His primary aim was to shorten the lead time between order intake and cars rolling of the production line (Ohno 1988). To effectively discuss and achieve constant improvement, he invented the term *value*. Value is defined as any action or process that a customer would be willing to pay for. The *customer*, in this definition, is anyone who consumes a product or service. In a sense, value is created only as the customer gets the car coming of the production system. Poppendieck et al (2003) argues that the same applies to software development, i.e. designs and prototypes in a developing effort are not valuable until the customer enjoys the delivery of the new product (Poppendieck et al 2003). The lean philosophy also includes *waste*. Waste is anything that does not bring value into the process. If the activity is wasteful, it should be eliminated as quickly as possible (Shingo 1988). When waste is eliminated, quality improves and lead time, production time and cost are reduced. In his work with TPS, Ohno identified seven areas of waste in manufacturing: *overproduction*, *unnecessary transportation*, *inventory*, *motion*, *defects*, *over-processing* and *waiting* (Ohno 1988).

Recent years have seen a growth of lean being applied to IT. In traditional lean manufacturing, the businesses produce goods of value to customers. In lean IT the IT function manufactures business services of value to the parent organization and its customers. The IT business services include, just like in manufacturing, resource management, quality control and security issues. The event of inventory being stored in the factory and the intermediate steps of an IT development process are not that different (Waterhouse 2008). Others argue that software development is a *creative process* rather than a well-defined production process, and as such, it should not be limited by the boundaries of what actually brings value to the customer (Bjørnvig et al 2010; Poppendieck et al 2003). There is, however, as both Bjørnvig et al (2010) and Poppendieck et al (2009) argue, no contradiction between having a creative process and a working framework to handle change. On the contrary, lean supports individual thinking to a strong degree, encouraging improvement by everyone involved in the working process. See table 2 for an comparison how lean manufacturing principles and agile principles differ.

Table 2. The contrast between lean manufacturing and agile software development in terms of core philosophy. (Source: Bjørnvig et al 2010)

Lean Manufacturing	**Agile Development**
Thinking and doing	Doing
Inspect-plan-do	Do-inspect-plan
Feed-forward and feedback (design for change and respond to change)	Feedback (react to change)
High throughput	Low latency
Planning and responding	Reacting
Focus on process	Focus on people
Teams (working as a unit)	Individuals (and interactions)
Complicated systems	Complex systems
Embrace standards	Inspect and adapt
Rework in design adds value, in making is waste	Minimize up-front work of any kind and rework code to get quality
Bring decisions forward (Decision Structure Matrices)	Defer decisions (to the last possible moment)

2.3.2 Lean Software Development

Now, with a basic understanding of both agile software development and lean principles, the next section is focused on the combination of the two in *lean software development*.

The most central aspect in lean software development is the concept of *waste*. Poppendieck explains the concept simply as *everything not adding value to the customer*, which is similar to Shingos (1989) original definition. Waste is, strictly speaking, the exact opposite of value. In software development it includes, but is not limited to, *unnecessary code and functionality*, *delay in the software development process, unclear requirements*, *bureaucracy* and *slow internal communication*. In order to identify and eliminate waste, one must be able to recognize and see it. If some activity can be bypassed, or the result can be achieved without it, the activity is waste (Poppendieck et al 2010).

This paper uses a tool called the *value stream map* to identify waste. The value stream map is an analysis tool which reveals wait states in any process. By analyzing the activities that take place in the chain from change request back to user, time spent working adding value to a product and time spent not adding value to a product can be mapped. When the map is complete, working time can be divided with wait time to provide a process efficiency estimate in terms of time spent in value-adding activities (Poppendieck et al 2003; 2010). As an addition, this workflow analysis can act as a base for discussing the *seven wastes of software development*. The following section will discuss them in greater detail.

2.3.3 The Seven Wastes

The seven wastes of software development (see table 2) is translated to the software development domain by Poppendieck et al (2003). This classification serves as a guide into the common wastes of software development processes. This section discuss and present Poppendieck et al (2003) classification, with supplementary observations by others researchers.

Table 2. The seven wastes as described originally in the TPS, and the translation into the software development domain. (Source: Poppendieck et al 2003)

Lean Manufacturing	**Lean Software Development**
Inventory	Partially Done Work
Extra Processing	Extra Processes
Overproduction	Extra Features
Transportation	Task Switching
Waiting	Waiting
Motion	Motion
Defects	Defects

Partially Done Work

Having software products partially done is risky (Poppendieck 2003). The longer the project remains unfinished, the larger the risk is of the product becoming obsolete. Poppendieck et al (2003) argue that a product needs to be put online in a production environment as quickly as possible for the proposed value to exist. It is impossible to know for sure what problems may arise with integration, or if the business problem the application aimed to solve really is solved. To have a project latent for a long time and then take it down is extremely wasteful. Partially done work also tie up resources (Poppendieck et al 2003).

Lundgren (2010) argues that defining *done* is essential in any development effort. Not only must done be defined on a functional basis, but also at an all compassing level, making sure that the product has quality built-in to it before being released to production. Lundgren points to several unprecedented cost and waste scenarios that may become real when done has an unclear or varying meaning among teams in the development organization:

- Products may not be sufficiently quality controlled, e.g. testing is partly skipped
- Solutions may not be cleared with the customer
- Documentation may be skipped

The definition should require a number of criteria to be fulfilled. For a development effort, done may be defined as: *coded* (implemented), *delivered* (checked-in, installed, etc.), *accepted* (by the product owner), *declared* (as in documentation) and *tested*. With a clearly defined done, the organization can quickly move tasks to the next workflow state or pull new work into the process. Finding useful done criteria is an iterative cycle that may take time. Kniberg et al (2010) simplifies the process somewhat by also setting definitions of done on each workflow state on a Kanban board (see figure 8).

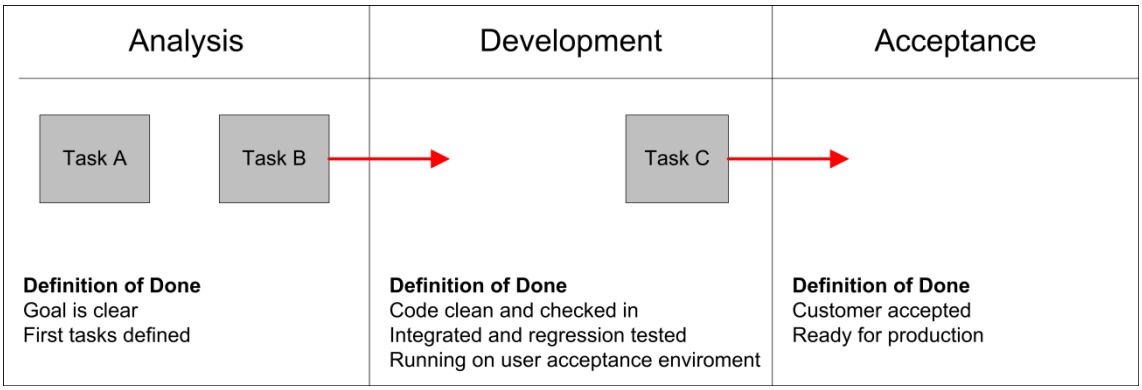


Figure 8. A Kanban board with three workflow states, including a definition of done criteria set. Task A and B, as an example, has fulfilled the done critera of the workflow state and is moved into the next. (Source: Author, modified from Kniberg & Skarin, 2010)

Extra Processes

Extra processes, as in papers that no one read, are waste. Poppendieck et al (2003) argues that paperwork

- Consume resources
- Hide quality problems
- Slows down response time
- Degrades and become obsolete
- Is lost

The software development team must always take into consideration the kind of paperwork needed. Common sources of waste are *change approvals*, *customer sign-offs*, and *traceability paperwork*. A useful test, according to Poppendieck et al (2003), is to simply see if anyone asks for the documentation being written. The creation of use cases, templates and tables that other members in the team are eager to use usually means that the papers are adding value. Poppendieck (2003) states three rules that, they argue, are useful to keep in mind if you need to produce paperwork:

- Keep it short
- Keep it high level, i.e. do not go into unnecessary detail
- Do it offline, i.e. on whiteboards or paper notes

Extra Features

Poppendieck et al (2003) describes extra features as unnecessary. The addition of code "just for the sake of it or because it's fun" (Poppendieck et al 2003) is pure waste. They argue that extra features, even the smallest edits in code, will force tracking, compiling, integration and testing to re-run again and potentially cause future problems. The Standish Group reports that an average of 64 percent of all functionality in software applications is never or rarely used (see figure 9).

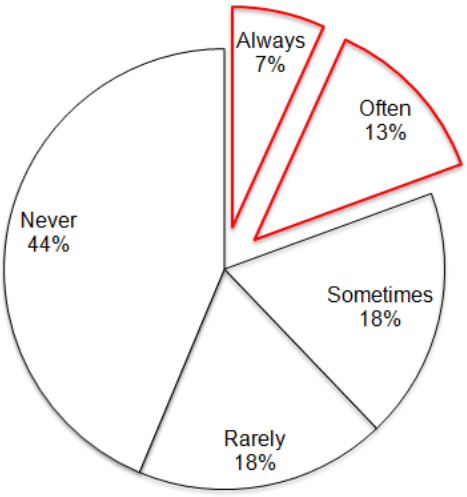


Figure 9. Chart showing the usage of functionality in an average application. 64 percent is never or rarely used, and only approximately 20 percent is used often or always. (Source: Standish Group 2002)

The low degree of functionality usage was particularly apparent in application where big designs up front were completed before business return-of-investment was considered or even conceivable. (Standish Group 2002)

Task Switching

Task switching is a common source of waste, according to Poppendieck et al (2003). In a task switching scenario, the following holds: It will take longer before *value can delivered to the customer*, and due to the attributes of *context switching* and *flow*, the same tasks will take approximately 20 percent more time to complete (see figure 10).

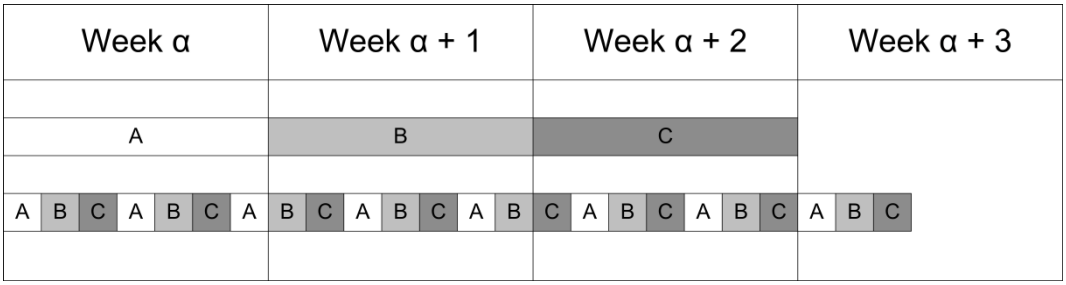


Figure 10. An comparison between task handling scenario 1 (sequential) and 2 (task switching). Task A delivers to customer A, task B to customer B and task C to customer C. In scenario 1, value is realized after every week. In scenario 2, only part of the needed work on each task A, B and C is done every week. As such, it takes three weeks for any task to deliver value. Moreover, because the developer has to change context frequently, approximately 20 percent more time has to be added, thus stretching the work into week 4. (Source: Author, modified from Matt Stine, 2010)

DeMarco et al (1999) describes the effects of context switching in software development. They use the term *flow* to denote a state where the developer becomes extremely productive. To frequently be able to reach this state, DeMarco et al (1999) and Shore (2004) points to several important prerequisites:

- If you *must* work on multiple projects, work on one at a time. Minimize context switches to maximize productivity

- If your team is required to handle interruptions (such as support events), consider rotating the responsibility for handling them. Make sure that each person has the knowledge necessary to handle the interruptions; otherwise the system will quickly break down
- Eliminate unimportant work and interruptions. If it isn't delivering value, stop doing it
- Ensure that the knowledge necessary to complete an assigned work task is in the hands of the developer. This will prevent the need to switch tasks caused by missing information (DeMarco et al 1999)

Waiting

Poppendieck et al (2003) argues that waiting is one of the biggest wastes in software development. Waiting is a holdup somewhere in the process, e.g. delays in starting a project, excessive requirement documentation, delayed deployment or unnecessary meetings. Delays are common in software development, and as such, it is common to think of it as acceptable. In essence, waiting is waste, as it increases lead time (Poppendieck et al 2009). The longer the lead time is, the longer it takes before the customer can benefit from the products of the SDP.

Motion

Motion is essentially the time it takes for a developer to find the answer of a question. The questions may be of a technical character, or in regard to customer demands. In any case, motion is about *reducing the effort, time and context switching it takes for a developer to become informed in a particular area* (Poppendieck et al 2003).

In the purpose of reducing such time, both Lundgren (2010) and Poppendieck et al (2003) argue the necessity of self-organizing, cross-functional teams. A cross-functional team requires a wide range of information to reach decisions, as opposed to a traditional functional team where decision making flows in a top-down fashion. Intra-team dynamics tend to become multi-directional rather than hierarchical. Processes encourage consensus within teams. Also the directives given to the team tend to become more general and less prescribed.

Corporate-level objectives drive business unit objectives, and functional departments rely upon decisions from business unit management. Poppendieck et al (2003) argue that organizations are becoming flatter, and functional departments are becoming less relevant. This due to the inherent waste in *motion* and *handovers*. Handovers between functional teams requires much communication and quickly becomes unsuitable for efficient work flows. A handover cannot possibly include all the tacit knowledge that has been accumulated over time. If a defect is found after the handover, teams or individuals involved before the handover may later-on have to be disturbed in order to fix the problem, thus losing their pace on current work tasks. Research has shown that at least 50 percent[2] of the accumulated knowledge is lost in handovers (Poppendieck et al, 2006). There are two scenarios which are particularly difficult, according to Lundgren (2010):

[2] 50 percent is a conservative number (Poppendieck et al 2006).

- Handovers of requirements and expectations from customer to technical implementation
- Handovers of code from implementation to test

If written handovers cannot be avoided, they should always be mixed with oral communication (Poppendieck, 2010).

Defects

The potential waste from a defect varies with *grade of impact* and *time until discovery* (Poppendieck et al, 2003). The longer the defect remains in the code, the more difficult it becomes to remove it. Research by McConnell (1996) suggests that a defect that is left undetected until construction or maintenance will cost fifty to two-hundred times as much to fix.

Management Activities

Poppendieck (2003) argue that management only holds indirect value in a software development process. It is important that managers have an understanding of the work they manage. Poppendieck (2010) argues that, without a technical background, managers are not in a position to provide guidance to technical workers. From a lean perspective, the fundamental job of managers is to understand how the work they manage works, and then focus how to make it better.

Control and tracking systems is usually a sign of waste in the process (Poppendieck et al 2003). When lead time is short, there is no or little need to keep track of projects in progress. The more projects running at the same time, the more task switching and waiting will occur among the members of the project team. The principles of flow in lean commend that an efficient workflow would require unfinished work in the pipeline to be kept at an absolute minimum (Poppendieck et al 2003).

3. Software Development at W&WW

The purpose of this chapter is to see how an enterprise company with a long tradition of industrial manufacturing works with software development. The chapter has the character of a case study, where how and why W&WW works with development is in focus. The disposition of the chapter follows the hierarchical structure of W&WW:

- Section 3.1 discusses the general characteristics W&WW and its relation to the parent company, ITT, which is the head of ITT conglomerate
- Section 3.2 focuses on the W&WW IT function and its relation to other business units within the conglomerate, as well as how the function itself is organized
- Section 3.3 present the software development teams' purpose and how they work to achieve this purpose
- Section 3.4 discuss the frameworks affecting the SDP, namely ITIL and PULS2 project management
- Section 3.5 gives insight into the SDP, starting from customer initiating RFC and ending with the RFC being implemented

3.1 Overview

W&WW is a part of the ITT Corporation conglomerate, a globally diversified manufacturing company with revenues of US 11.7 billion in 2008, and over 40 000 employees (ITT official site 2010). ITT participates in several other global markets, including defense and security, and motion and flow control. ITT is organized as three *business divisions*, with a number of *value centers* as subsets of each business division (see figure 11). In the ITT organization, every value center has a certain degree of autonomy. Recently, however, a corporation-wide project called "One ITT" has made efforts to centralize and unify the company, with the purpose of closer cooperation between the parent company and the value centers (W&WW intranet 2010a).

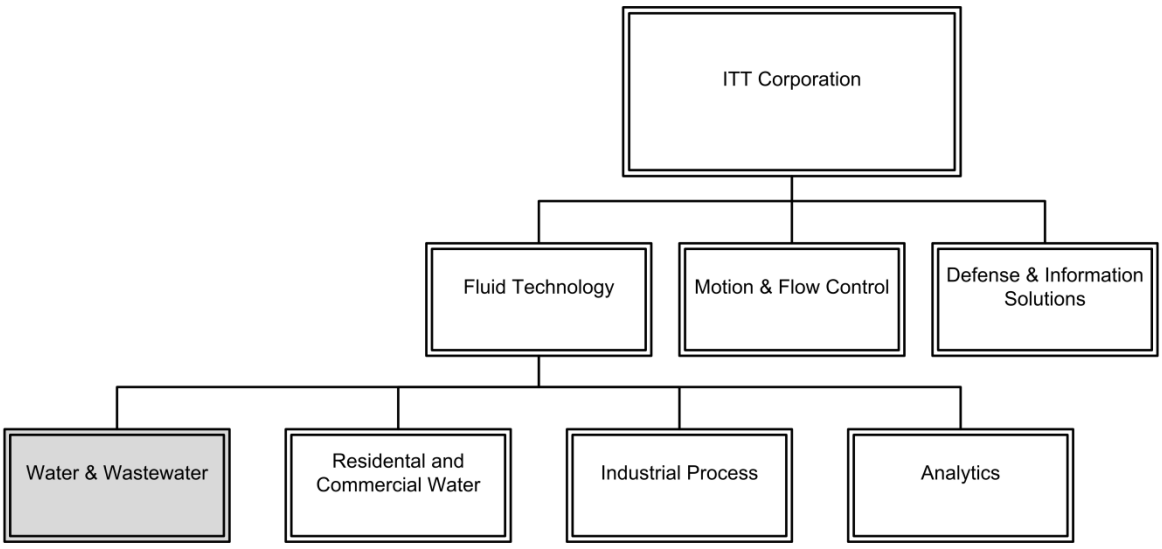


Figure 11. Organizational overview of the ITT conglomerate. On top is the ITT conglomerate head office. On the second level are the business divisions. The third level is value centers, including W&WW. (Source: Author, based on W&WW intranet 2010b)

The Fluid Technology division is the world's largest provider of water and wastewater treatment solutions and a leading provider of pump and related technologies for

industrial, commercial and municipal customers. The business division had revenue of approximately US 3.5 billion dollars in 2009 (ITT Fluid Technology 2010). W&WW is organized as a subset of Fluid Technology. With approximately 5 800 employees globally and 1 500 in Sweden, W&WW manufactures solutions for water and wastewater transport, biological treatment, filtration and disinfection. W&WW started out as a small forge in Emmaboda 1901 but has grown steadily over the years. In the beginning of 2000 it was acquired by ITT and organized under the Fluid Technology division (Teamleader A 2010). The headquarters are today in Sundbyberg, with R&D and marketing functions organized there, while distribution, manufacturing and finance functions are still in Emmaboda. Functions such as IT, human resources and purchasing are represented at both locations to some extent. W&WW also has smaller manufacturing plants spread out globally, namely in Shenyang, Buenos Aires and Herford. (W&WW official site 2010)

3.2 IT function of W&WW

The IT function, with a staff of approximately 150 employees, is distributed over two locations: the headquarters in Sundbyberg, which has the larger part, and the manufacturing plant in Emmaboda, where some IT related services, mostly in connection with software development, still takes place. (Teamleader A 2010; Teamleader D 2010)

The IT function is arranged in *teams*, with approximately half of them focusing on IT operations and the others on software development. Part of the IT operations, the helpdesk and application management teams, were recently moved to a centralized IT function in the Fluid technology division called the *Service Delivery Organization* (STO) (Teamleader A 2010). The STO employs client technicians, server technicians and other roles directly related to the IT infrastructure and operations. They do not belong to W&WW organizationally, but their services are hired by W&WW (Teamleader A 2010).

The other part of the IT function is the software development organization. They are responsible for in-house development of tools related to selection, calculation and design related to the offerings of W&WW. Almost all developed software applications are for internal use, i.e. use within W&WW. All software development is – in contrast to helpdesk – still directly organized under W&WW. A reason for this is that W&WW consider software development "valuable as a differentiator against the competition [as] most of them simply cannot offer the services we offer" (Teamleader A 2010).

3.3 Teams Involved in the SDP

The IT function is split into three branches, *departments*, with one to three teams organized in each department (see figure 12).

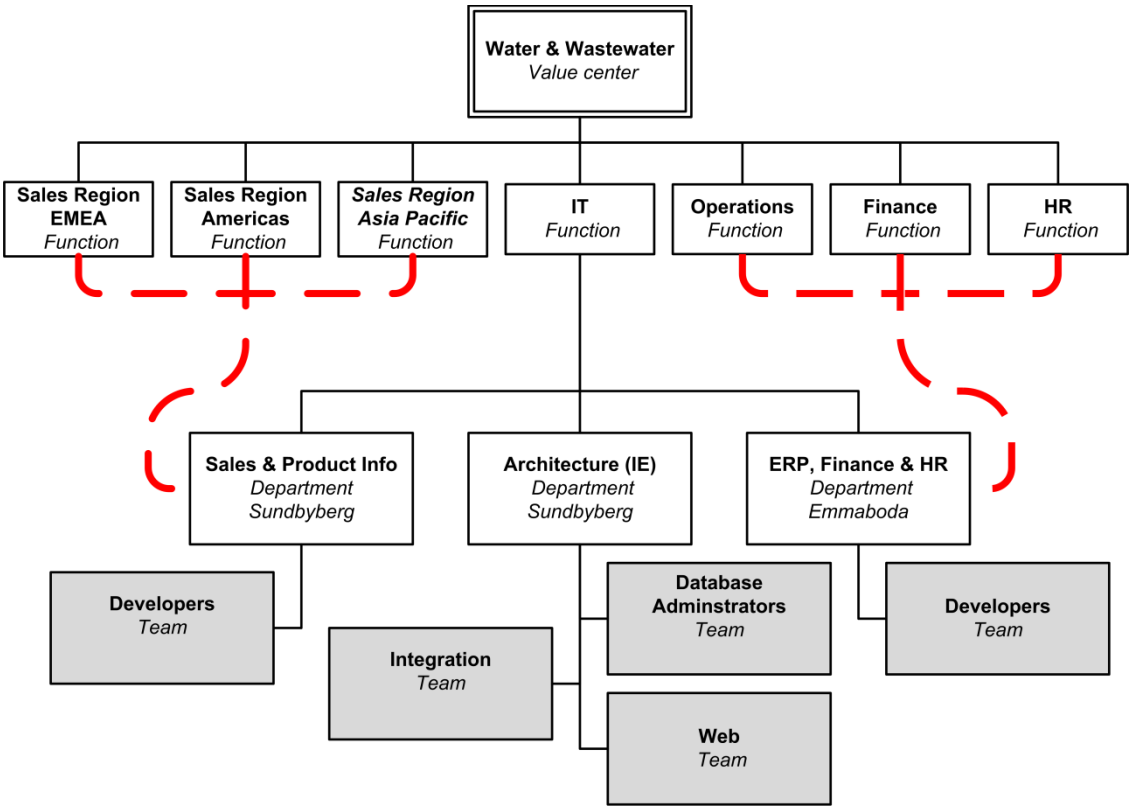


Figure 12. The software development teams of W&WW, in a combined organizational and interactional view. Under the W&WW head is an umbrella of functional units. The software development teams are organized under departments subordinate to the IT function. Grey boxes symbolize teams directly involved in the SDP, and red, crosshatched lines symbolize interaction between departments (offering software development) and functions (acting as customers). (Source: Author, based on W&WW Intranet (2010))

The development effort is based around *teams*; function-specific groups with one *team leader* and three to fifteen members, often geographically as well as organizationally disperse from one another. Five separate teams are identified: the *database administrator team*, the *web team*, the *integration team*, the *Sundbyberg developer team* and the *Emmaboda developer team*. Both developer teams are large, including more than ten members each. All teams belong to a department in the organization. The departments are structured in accordance with the customers, i.e. the

- Emmaboda developer team, belonging to the *ERP, Finance & HR* department, serves the *Operations*, *Finance* and *Human Resources* functions. This department is located next to the manufacturing plant in Emmaboda
- The Sundbyberg developer team, belonging to the *Sales & Product Info* department, serves *R&D* and *sales* functions. This department is located at the headquarters in Sundbyberg
- The third department, called *Architecture*, is responsible for IT architecture, with teams administering databases, integration solutions and web milieus (Teamleader E 2010)

The current setup of teams, aligned to function, is "relatively new" (Teamleader E 2010). Earlier they had setups with a project manager and a mixed set of developers in one small team, each catering to one or a very few specific customers. The change to

functional teams only came recently, with the primary reason being to address the inflexibility that came with having such strong ties to one or a few customer (Teamleader E 2010). The change also suited the ITIL IT operations framework well (Teamleader B 2010).

3.3.1 Emmaboda Developer Team

The developer team in Emmaboda belongs to the ERP, Finance & HR department. The team consists of one part dot-net developers and one part COBOL developers. The dot-net developers are four people, mainly developing for web interfaces, while the Cobol developers primarily develop for the mainframe using a system called IDMS. The applications being produced are mostly used in the manufacturing and operations side of the company, including plants in USA and Germany. (Teamleader F 2010)

Among the products being developed by the Emmaboda developer team is E-purchasing, an Internet purchasing tool for suppliers. The portal is mainly used for electronic communication between W&WW and its suppliers. It was launched during the fall of 2002 and has been under continuous development since. Over the years it has earned recognition and been given the ITT Ring of Quality Award and a nomination as the Project of the Year by CIO Sweden (Developer A 2010). During 2008 to 2010, the ambition is to implement E-purchasing as the sole ITT supplier portal world-wide (W&WW Intranet 2010b).

Methodology

In 2008 the team begun experimenting with a Scrum-based methodology, initiated by one of the newly hired developers who joined the team. He had used Scrum in his previous workplace and had both appreciated and enjoyed this way of working (Developer A 2010). They used the book *Scrum and XP from the Trenches* by Henrik Kniberg (2007) for guidance, and "followed it pretty much to the letter" (Developer A 2010). The teamleader was given the Scrum Master role. She had no previous knowledge of what it actually meant or what she was supposed to do, but it "quickly became clear". (Scrum Master 2010)

3.3.2 Sundbyberg Developer Team

The Sundbyberg developer team consists of six people – a mix of employees and consultants – geographically placed together at the same floor of the headquarters. The customer group is mainly R&D, sales and marketing, all located in the headquarters as well (Teamleader E 2010). The team supports over 40 systems and applications which range from "practically un-used to business critical" (Teamleader E 2010). The application portfolio include *configurators*, which are systems to assist in the selection of equipment for a particular distributor, and *documentation centers*, which include documentation and service guidelines that the distributors can access directly via the web. The configurators are the area where most effort will be put in the future (Teamleader A 2010). They differentiate W&WW from the competition by offering a "powerful yet cheap" (Teamleader A 2010) way to calculate and simulate different scenarios, simply by contacting the sales function. Other companies does not offer this service, instead they force you to use specialist consultants in order to get the proper calculations for your equipment (Developer B 2010).

The main technical platform is dot-net, but because the team also administers wide range of old systems built over a time-span of 30 years, there is also a need to maintain

skills within FORTRAN and Visual basic for maintenance work (Teamleader C and D 2010).

Methodology

Only parts of the development effort are organized according to any methodology, and those parts are mostly routines related to ITIL. Due to a large number of applications and the specialist knowledge they require, much of the development is instead run on an ad-hoc basis by single developers (Teamleader A 2010). When there is a lack of competence, consultants are often contracted. The consultants, however, does not "bother which methodology they are using, they just work as they are told to" (Teamleader A 2010).

3.3.3 Integration Team

The integration team consists of one teamleader and three consultants, all localized next to one another in the headquarters. The purpose of the integration team is to offer "flexible integration solutions" (Teamleader B 2010); i.e. to make communication between IT systems efficient. In practice this is done by exposing services and APIs for other systems to access. By keeping the system dependency low, it's easier to make changes in one system without affecting the performance of other systems (Teamleader B 2010). The team updates old as well as new systems with services that are constructed in accordance with this principle.

Methodology

The team is overloaded with requests from other teams which sometimes lead to long wait times for tasks entering the integration phase (Teamleader B 2010; Teamleader E 2010). In order to reduce wait time and keep better track of current work items, they use a digital Kanban board. The board (see figure 13) is stored in a collaboration space on the intranet. Part of the purpose of keeping the Kanban board digital is to give other teams – and "anybody else who is interested" (Teamleader B 2010) – some insight into what the team is currently doing, and what they are planning to do for some time ahead. (Teamleader B 2010)

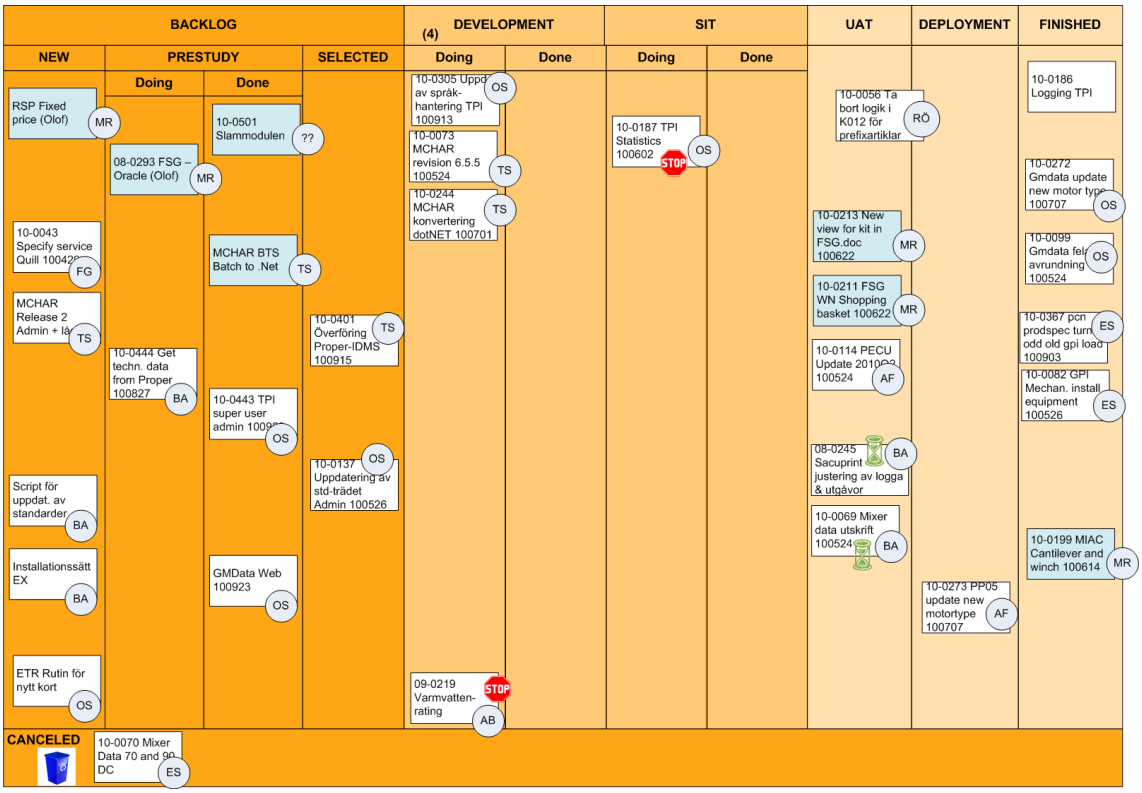


Figure 13. Digital Kanban board, as used by the integration team. Tasks are picked from the backlog state and then moved to the right as coding, testing and deployment takes place. Notice the red stop signs on some tasks, indicating that the task is waiting for input. (Source: W&WW Intranet 2010c)

Because so many tasks are waiting for input, the board is currently plotted with stop signs. When all tasks have stop signs, the team simply adds more tasks, ignoring the work-in-progress limit for the current workflow state (Teamleader B 2010). The board has helped for internal team coordination and visibility, but it does not help coordination with other teams. There is still too much external dependencies to other teams to make the work flow efficient and lead time shorter (Teamleader B 2010). Neither does the other teams prioritize nor "select tasks the same way we do" (Teamleader B 2010). The problem of coordinating work between teams has been noted several times, but discussions with management have not corrected the problem. The integration team has the mandate to change their work process, but not intervene with how other teams work (Teamleader B 2010; Teamleader E 2010). All in all, however, the team is satisfied with their Kanban board and thinks that it could serve as a first step to facilitate coordination if the other teams had one as well (Teamleader B 2010).

3.3.4 Database Administrator Team

The database administrator team is responsible for the design, implementation, maintenance and repair of W&WWs databases. Also included in the tasks of the team are development and design of database strategies, monitoring and improving database performance and planning for future expansion requirements. As such, all software development efforts must seek acceptance with the database administrators before deploying to the production environment (Teamleader A 2010). Although database models and relations can be constructed by the developers themselves, they must still

seek acceptance from the database administrator team to deploy any changes (Scrum Master 2010).

3.3.5 Web Team

The web team has, similarly to the database administrator team, a primarily operational role in the organization. They are not directly involved in the software development effort by means of coding; instead they have the responsibility of uploading and ensuring the continuous viability of applications made available on the intranet, the extranet and the Internet. (Teamleader A 2010)

Methodology

The team picks tasks using a helpdesk tracking system. Tasks are delegated from the main helpdesk team belonging to the service delivery organization (see 3.2). Once delegated the task shows up on the screen of the team member. The member then picks a task, often related to her skills or a priority level. (Teamleader A 2010)

3.3.6 Communication and Coordination

There is little coordination in terms of how the developer teams approach their daily work, and long-time cooperation is rare (Teamleader F 2010). The teamleader in Emmaboda could only recall one project in the last few years where they had cooperated. That project had a cross-functional setup, with developers from both the Sundbyberg and Emmaboda team working together. The team leader recall this project being a success, and that it was "wonderful to have all the needed competences available in the project under one manager" (Manager 2010). The possibility of further cooperation in such manner has not been investigated. As of 2010, there is no widespread knowledge in regard to how the other department is working – or even what they are working with (Teamleader F 2010).

From a coordination viewpoint, tasks are generally carried out in a sequential fashion, i.e. once a team is done with their part, the task is moved on to another team. Practically all tasks need expertise from at least two teams (Developer A 2010). The Emmaboda developer team has noted extensive lead-times when interacting with other teams to test or integrate (Teamleader F 2010). Even if they succeed with their Scrum sprint, i.e. produce deliverables according to plan, the other teams might not be ready to take up the task. This often because different teams may have different work in the pipeline, and that work may have been prioritized higher by the department manager. On occasion, wait times are several months (Teamleader F 2010).

There have been at least two efforts to improve the communication. A few years ago, a resource planning meeting was held regularly. In the end, however, the meeting became "useless" (Manager 2010) because all managers could not account for what resources they disposed and when the resources were available (Manager 2010). Currently, a new effort is taking place to coordinate the teams. This time it's the teamleaders working together to schedule meetings where some basic coordination can be made (Teamleader E 2010). The meeting is usually short, mainly just asking "what are you guys doing the next few weeks?" (Teamleader B 2010). The effort has not yet been evaluated, but there is a general positive attitude towards the initiative (Teamleader B 2010; Teamleader E 2010). The meeting, however, is limited to the Sales & product info department. Teamleaders in other departments have not been involved in the effort (Teamleader E 2010).

3.4 Frameworks

The Software development teams are subordinate to the IT function, and the IT function is subordinate to the IT function of the parent company, ITT. As such, some IT policies and practices that the parent company adheres must also be by honored by W&WW. Two frameworks affect the development effort directly (Teamleader A 2010):

- *Information technology infrastructure library* (ITIL), a framework of "best practices" and concepts for IT functions and, particularly, IT operations. ITIL is the basis of the processes describing software development (mainly the *change management* process)
- *Value-based lean six sigma* (VBLSS), a lean-based philosophy derived from the manufacturing and production part of the company. The project management model PULS2 is maintained by the VBLSS function

They both affect the routines and activities of the SDP, and as such they must be taken into account as important factor regulating the process.

3.4.1 ITIL

ITIL is a set of concepts and practices for IT service management. Its main focus is to

> *Optimize the operations of an IT organization, i.e. the processes responsible for the day-to-day monitoring and management of IT services and the IT Infrastructure they depend on* (Richmond Systems 2010)

As an IT service management framework, ITIL is more concerned with the above mentioned operational concerns of the IT function, and less with technology and software development (ITIL official site 2010; Teamleader A 2010).

ITIL is very prescriptive. It includes detailed descriptions of all common practices in IT. The descriptions provide wide-ranging checklists, tasks and procedures that "pretty much any IT organization can tailor to its needs" (ITIL official site 2010). ITIL is published in a series of books, each of which covers an IT management topic. The United Kingdom's Office of Government Commerce is the organization behind the framework, and they continuously updated the framework to match lasting trends in the industry. (ITIL official site 2010)

Over the years, ITIL has shifted management approach several times. Early on it described IT as a pure support function. In 2001, process management was added. In 2007 ITIL shifted focus to include IT as a value-adding business, publishing it as version 3. Today the framework has a service-oriented focus, with the purpose of integrating IT with the core business of the company. It has also become the most widely used IT service management approach in the world. (ITIL official site 2010)

Change Management

The ITIL Change management process, which is central to the W&WW SDP, is about *managing change*, i.e. assessing the impact, cost, benefit and risk of a proposed change in the IT environment. At ITT W&WW this explicitly involves developing business justification and obtaining approval, managing and coordinating the change implementation, monitoring and reporting on implementation, reviewing and closing change requests (Teamleader E 2010). ITIL defines change management as a way to

> *[…] ensure that standardized methods and procedures are used for efficient and prompt handling of all changes, in order to minimize the impact of change-related incidents upon service quality, and consequently improve the day-to-day operations of the organization* (ITIL official site 2010)

Change management processes are often responsible for managing change in hardware, communications equipment and software, system software, and all documentation and procedures associated with the running, support and maintenance of live systems (ITIL official site 2010).

ITIL in projects

At ITT W&WW, change management is not responsible for overseeing changes that occur within development projects. Projects are delegated to a change management process dictated by the project management methodology called *PULS2* (W&WW Intranet 2009). Still, close communication between project managers and the change manager is a must, as the project manager may be required to utilize change management resources for tasks that need the production or test environments, i.e. testing or release tasks (W&WW Intranet 2009).

3.4.2 PULS2 Project Management Template

The project management template PULS2 is used by all units except R&D at W&WW (W&WW Intranet 2009). The purpose of the PULS2 template is to offer a set of specific plans and actions to achieve a change in a given timeframe with focus on cost and scope constraints and to utilize resources effectively. The PULS2 template offers a set of activities and toll gates (see figure 14).

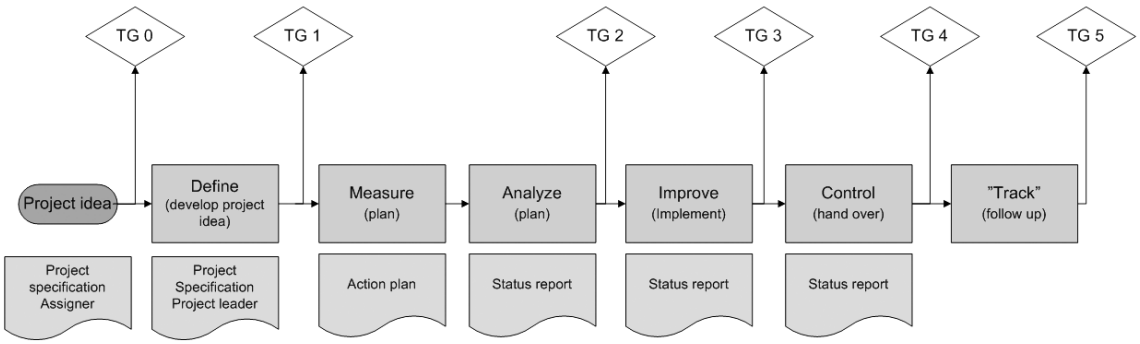


Figure 14. Project Management PULS2 template. The template includes six gates that must be passed before the product is deployed: TG 0, TG 1, TG 2, TG 3, TG 4 and TG 5. (Source: Author, based on W&WW Intranet 2009)

The purpose of the toll gates (TGs) is to secure proper completion of the previous step, thus verifying that the project is ready to move on to the next phase. TGs are explicitly described as a risk assessment tool to "lower the risk of the project" (W&WW Intranet 2009). To pass each gate, the requirements are as follows:

- TG 0 requires project prioritization and an assigned project leader
- TG 1 requires a go-decision to start the project. There must also be a project group and decision group (the decision group take the TG1 decision)
- TG 2 requires a decision regarding which solution to implement. After TG2 the development start

- TG 3 requires a final implemented solution. A decision must be taken on what is needed for a handover to the project receiver/customer. This tollgate is regarded particularly important, because it include the formal (and mutual) decision to handover the solution to line management. The control document explicitly state that "many projects have previously ended here due to a vague hand-over process and expected projects have not been fulfilled." (W&WW Intranet 2009)
- TG4 requires verification that the hand-over to the customer is finalized, and performed in a satisfactory way
- TG 5 is a follow-up evaluation of the project. This part is almost never completed (Teamleader A 2010)

Since PULS2 is not part of the change management process of the ITIL framework, it is instead layered over the change management process. As such, it adds or extends some activities in the SDP. Among these activities is the *project application request* (PAR) revision. The purpose of the PAR is to secure the economical validity of the project (W&WW Intranet 2010d). PULS2 requires that all projects complete a PAR revision before initiated.

3.5 The Software Development Process

The SDP is for the most part, as we noted in 3.4, described in the *change management* process. W&WW has made an effort to map the entire SDP in a set of change management process maps. Depending on the work form, task or project, of the software development effort, the SDP may either be limited to the

- change management process (see Appendix III) or
- change management process combined with the formal PULS2 project management template (see Appendix IV)

The change management process map is a few years old, and the PULS2 process map is only in draft release (and has been so for a year or more). Therefore the actual workflow might be somewhat different to what the process maps suggest. They, however, serve as a guide to structure the workflow in this section. From a high level perspective, the SDP can be merged to one model, both for tasks and PULS2 projects. The whole process then includes six activities (see figure 15).

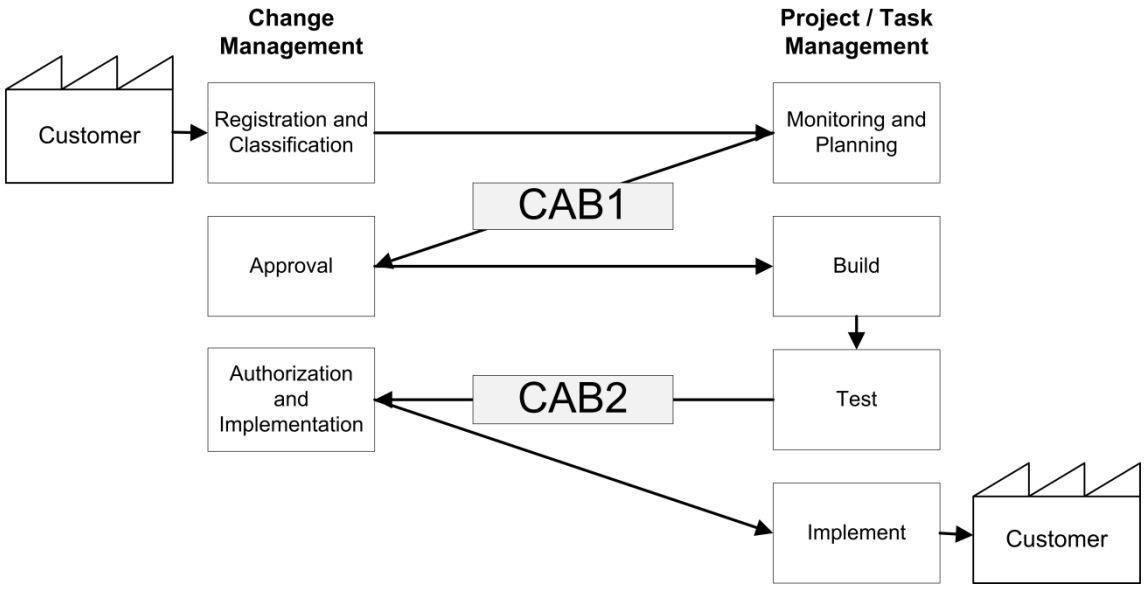


Figure 15. The SDP as an interaction between the change management process and the task and project management model. The change advisory board (CAB) serves a gate keeper between steps in the process. (Source: Author, based on Rudy Stubler 2008)

The next section goes through the relevant activities, starting from the *customer initiating an RFC* and ending with *implementation*.

3.5.1 Customer Initiating an RFC

Every development effort is started with an idea or suggestion. The idea can be an improvement or bug fix to an existing system, or be an idea for a completely new system. The idea is formalized in a *request for change* (RFC). Within W&WW ITIL terminology, RFCs contains a *description of a wanted change in an IT system* (Teamleader A 2010). The RFC states what is in need of change, but does not include how the change is to be carried out. The attributes includes an *ID*, *name* of the customer ordering the change, *deadline* for the change, an indication whether the change is required or optional, a *change type* and an *abstract* (see figure 16).

ITT

REQUEST FOR CHANGE, RFC

ITT Water & Wastewater AB

Page: 1 (6)

Subject *	Handler/Project Manager *	IFS account *	RFC No *
Fler decimaler i Kvantitets fältet			

Mandatory = *

Fill in by the Requester > Send the RFC-document to the Customer Responsible at IS/IT department

Requester*:	Date*:
[redacted]	2010-11-02
Application / program/Database*	Requested ready date:
trcukkön	2010-12-01

Problem description/Project idea – Please attach possible investigation documentation or any other documentation of interest (it may serve as the basis for decision)*

Truckkön tar enbart emot 1 decimal I antal fältet, minst två decimaler måste tas emot

Impact Analysis - Business impact /value*

Scanning kan inte användas på t.ex. meter och kilogram

High prioritized request due to audit findings or other security assets. * Yes ☐ No ☐

Figure 16. A Typical Request For Change (RFC), filled in by a customer. The RFC includes fields for requester (customer), date/ready date (deadline), affected application/program/database, problem description and impact analysis. (Source: W&WW Intranet 2010c)

RFCs usually originate from one of the following five sources (Teamleader A 2010):

- Problem reports that identify bugs that must be fixed (the most common source)
- system enhancement requests from users
- events in the development of other systems, due to dependencies
- changes in underlying structure and/or standards (e.g. a new operating system)

The RFC is filled in by a *requester* – a customer or user – and then e-mailed to the inbox of the *customer responsible*. The customer responsible then analyzes the RFC, which typically takes 1 hour (Purchaser 2010). If it's deemed to need a more detailed specification a pre-study is conducted, commonly with the help of one or a few developers (Teamleader F 2010). The RFC is then returned to the customer for consideration and approval. If accepted, the RFC is *classified* during registration and classification.

3.5.2 Registration and Classification

Depending on the result of the pre-study, the RFC is classified as a *PULS2 project* (over 200 working hours and one million SEK) or a *task* (less than 200 working hours). In addition, W&WW control document (W&WW intranet 2010e) state that a project must have the following characteristics:

> *[...] unique and has well defined improvement objectives; is of non- recurring nature; has a temporary project organization; and is limited in time.*

If the RFC is classified as a project, project management methodology PULS2 must be followed. Projects using PULS2 are not very common with the Emmaboda developer team. They estimate that less than 20 percent of their time is spent in projects (Scrum Master 2010). In Sundbyberg, the developer team estimates that they spend 40 percent of their time in projects (Teamleader C 2010).

3.5.3 Monitoring and Planning

After classification and authorization, the RFC is registered as being ready for planning. It is then delegated to the appropriate developer team with regard to the customer. How the RFC is handled once it is delegated differs greatly between the teams. With the developers in Emmaboda, the RFC is stored in a spreadsheet (see Appendix II). In Sundbyberg, on the other hand, most RFCs are registered in a collaboration web portal used for tracking projects (see Appendix I). In a few days, when the handler is available, the RFC is up for resource planning (Teamleader E 2010). The purpose of resource planning is to claim resources so that the RFC task list can be completed. When time estimation and resource allocation is done, the RFC is sent back to the *program responsible* (Teamleader E 2010). The program responsible may choose whether to authorize and thus continue the process, or reject and stop the process. Should the program responsible choose to continue, then a *handler* must be selected. The handler is, essentially, a project manager in ITIL terminology (Teamleader E 2010). With the appropriate handler selected, authorization must be given by the system owner before the program responsible can update RFC with handler information. When the resources and the handler is deemed available, then the resource owner give a go to the handler. The handler then starts planning, which typically takes one or two days. When planning is ready, the RFC enter the next stage, CAB1 approval. Physical CAB1 meetings are only summoned once a month, but approval can also be attained via e-mail (Teamleader E 2010).

3.5.4 Change Advisory Board Meeting (CAB1)

The CAB meeting is a central activity in the W&WW change management process. It serves to balance the need for change with the need to minimize risks of data loss or downtime (Teamleader E 2010). The CAB is responsible for overseeing all changes in the production environment. As such, it has requests coming in not only from IT, but also from management, customers and users (Teamleader E 2010). The changes involve hardware, software, configuration settings and patches (ITIL official site 2010). The CAB group delivers support to the change management team by approving requested changes and assisting in the assessment and prioritization of changes. Members are chosen to ensure that the requested changes are thoroughly checked and assessed from both a technical and business perspective (Teamleader E 2010). The RFC type will dictate the required personnel to participate in the meeting. They are not required to meet face-to-face on each requested change, but rather use electronic support and

communication tools as a medium. Once the CAB1 meeting is over and the RFCs are rejected or accepted and prioritized, the building and test phase begins (Teamleader E 2010).

3.5.5 Building and Test

If the RFC, in keeping with change management, is considered a *standard change*, then a *standard operating procedure* (SOP) is used. The SOP defines when, where, how, by whom and under what circumstances a standard change should be conducted. It is a written document detailing all the steps and activities of a standardized procedure. In the W&WW SDP, and probably most software development efforts, not many changes are considered standard changes (due to their complex nature, see 2.1.1). It is thus rare that a standard operating procedure is performed. The Sundbyberg teamleader could not recall "any occasion at all" (Teamleader E 2010). Instead, the development process typically sets in, where a software solution is built and tested from scratch. (Teamleader E 2010)

The development process involves coding and testing which, for an average task, typically takes typically 50 hours to complete (Teamleader E 2010). How these activities are organized differs between the teams. The Scrum-based Emmaboda developer team generates work items from a spreadsheet. With a built-in script, they can automatically create work items that correspond to RFCs. The cards then serve as the product backlog for the upcoming sprint meeting. Once the sprint backlog is set, the sprint begins and then carries on for four weeks. Recently, however, the time-box was reduced to two weeks in an effort to shorten lead time. The effects of this change have not yet been evaluated (Manager 2010). At the start of each Scrum sprint, as the developer picks a task, he immediately calls the customer to arrange a UAT. This routine enables testing to be done as soon as the code is considered ready. (Manager 2010) An issue experienced by the teamleader in Emmaboda is that RFCs often need to be split into two or more work items to be feasible to code within one sprint. If RFCs are too large when pulled into the sprint they may not be finished during the time-boxed iteration, rendering the burndown charts useless. (Teamleader F 2010)

The Sundbyberg developer team's development phase is handled differently. The developers are being delegated RFCs that suit their particular area of expertise, usually one or a few applications towards one or a few customers (Teamleader E 2010). The customer responsible group, who selects the RFCs to work on, does not prioritize work internally, although efforts to "have scheduled prioritization meetings is in the pipeline" (Teamleader E 2010). It's hardly ever more than one developer working on a task or project at the same time (Teamleader E 2010).

Added code to applications is not reviewed by other developers. Neither are there any requirements to follow code conventions or other standardizations. Most code branches are currently not stored centrally. Recently, however, an open-source code configuration tool, *Subversion*, was installed with the purpose to centralize all management and configuration of application code. The idea came from one of the consultants who felt that it was too complicated to overview the code base of all applications. (Teamleader E 2010)

When the build and test activity is done, the handler and system owner must approve the new solution, which typically takes a few days. If approved, the solution is ready for

deployment into production. The CAB2 meeting is summoned to approve the final release of the into production. (Teamleader E 2010)

3.5.6 Change Advisory Board 2 (CAB2) and Implementation

The purpose of the CAB2 is to authorize the deployment of solutions to the production environment. Depending on the size of RFC and the corresponding solution the meeting might be shortened to a few e-mails or phone calls. If the RFC is significant, however, a full-scale CAB meeting is needed. These are only held once a month, at a specific date. The change release is then fitted into a *slot*, a time-boxed window when changes to systems can be made. The new update or system is then available for the user. (Teamleader E 2010)

4. Analysis

The analysis part of this study is conducted in four steps:

- First a value stream mapping (see 2.3.2) of the entire SDP. The purpose of this analysis is to estimate efficiency of the process in terms of value-adding activities set against wait states. It also reveals the most immediate bottlenecks impairing lead-time in the process
- The second part focuses on the development (build and test) part of the SDP and uses the seven wastes concept as guidelines to qualitatively discuss waste in the SDP
- The third step merges insights from step one and step two and discusses how W&WW current agile implementations are performing in this environment
- The fourth part discusses and proposes changes in the SDP to better adapt it to lean and agile philosophies, and what the benefits and costs of such a change may be

4.1 Value Stream Mapping and Process Efficiency

The value stream map is a natural starting point in analyzing the efficiency of an organization (see 2.3.3) and serves as the starting point of the analysis. The purpose of the value stream is to capture work-time and wait-time in the whole process from customer back to user. A number of assumptions were made during the construction of the map:

- The value map is a high-level approach, as discussed in 2.3.2, causing some activities to be bundled together
- The unit of measurement is discrete hours, i.e. the minimum time spent on any activity is one hour
- When an activity is estimated with a time span, the wait-times and work-times are roughly estimated with the expectation-value, i.e. a four-to-six weeks task would take five weeks in average
- The teamleader was explicitly instructed to round of times down to the nearest hour. As such, all wait-times are probably somewhat longer in the real workflow
- The development phase (building and testing) is seen as one activity, although a more detailed investigation in the next section, 4.2, will show that waiting time occurs within this phase too

The final value map is presented in figure 17.

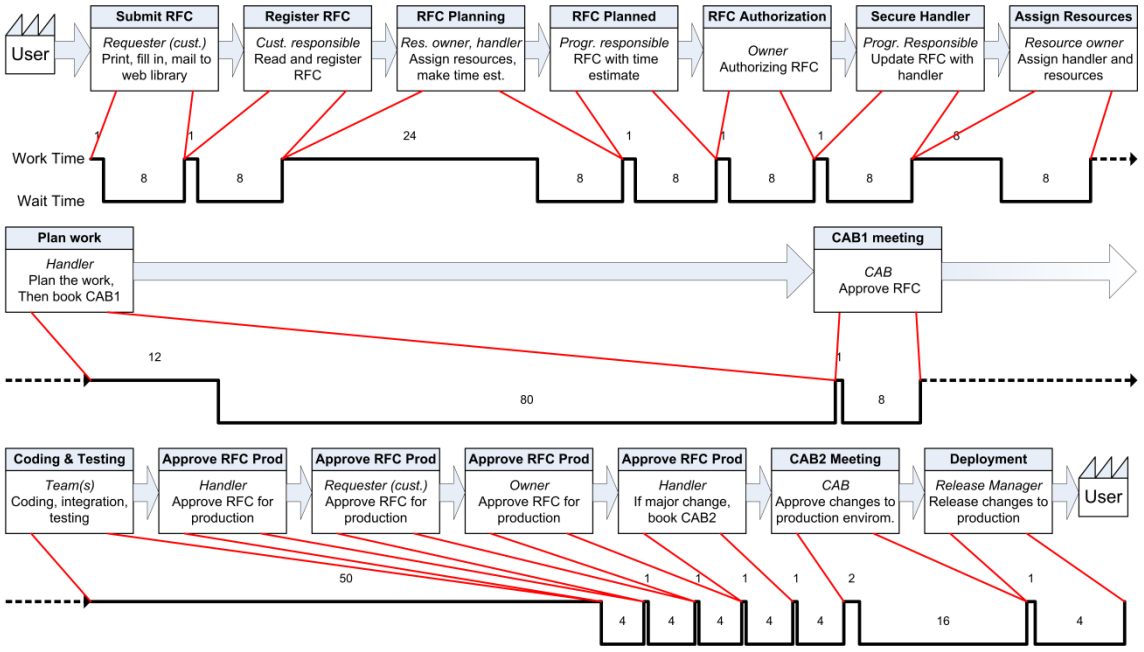


Figure 17. Value stream map of the SDP on three inter-connected rows, starting with a request for change and ending with deployment. The value map identifies work time and wait time for 16 activities in the process. (Source: Author)

The value stream has 107 hours of value-adding work time, out of a total of 300 hours in the process. Division gives that 36 percent of the process is value-adding, thus leaving approximately 64 percent as waste. A one-third efficiency rate is much in line with how prescriptive traditional SDPs usually perform (see section 2.3.3).

4.1.1 Bottlenecks

The most significant waste of the process is the wait-time for the CAB1 meeting. Because the meeting is scheduled only once a month, a task is prolonged with an average of 80 hours during this phase. A second prominent waste is the handshake actions; there is a systematic waiting time in confirming different steps in the process. Because so many actors and regulatory actions are required, much time is spent just filling in, mailing, printing and signing papers. A few of these steps seems to be approval processes that *always* grants approval as well. Such process preparation work and project close up work is very wasteful. There is no independent value to these activities, nor do they produce any direct or indirect value in the final product.

4.1.2 Waste

To go into more detail and identify waste *in* the activities, the seven wastes of lean software development is just for comparison (see 2.3.3). Not all wastes are easily identified in the context of W&WW; this does not make them non-existing. They can be hidden by paperwork or ad-hoc processes.

Partially Done Work

Partially done work is common in the W&WW organization, indicated by a large number of long-running and unfinished projects (see 3.5.2). The PULS2 template explicitly states that many projects never make it to TG 4, indicating that the problem is known and existing, but not addressed properly. In this case, the holdup seems to be

outside of the requirement and development phase (which is before TG 3). Ergo, the software development solution is finalized, but the organization cannot afford, or are not interested, in taking ownership of the product. These problems may be due to quality problems, i.e. that the product do not fulfill the expectations or demands of the customer, or that the project overran time and budget, and thus is not relevant anymore for the paying customer. Instead of paperwork such as the PULS2 template, with explicit instructions that aims to remedy inherent waste or quality problems, lean software development principles provide that these problems should be confronted much earlier. With minimized lead time and iteratively adjusted customer requirements, few projects should need to be cancelled or caught in a status quo at TG4.

Extra Processes

The e*xtra processes* waste is essentially about unnecessary processes that not add any direct or indirect value for the customer. At W&WW, the terminology and activities prescribed by ITIL is used in most IT processes. As such, it has a central and important role in the IT organization. W&WWs ITIL implementation has reached a state somewhere between version 2 and 3; it's transitioned into process management but has not yet implemented all service-oriented aspects of ITIL (see 3.4.1). Even though ITIL version 2 strongly advises the use of process-oriented workflows, few of W&WW process maps actually show up-to-date information of how the SDP works. In particular maps that describe the integration with PULS2 are non-existing. Work may be in the line for new process maps, but so far it's taken half a year and yet only resulted in drafts (another case of *partially done work*). It is also interesting to note that the specific development phase (as in build and test) not even has a process map. As of now, much of the effort is on an ad-hoc basis, with only a few managers who can overview the whole process and all actors involved. They are also the only ones with mandate to change it.

The combination of ITIL best practices and PULS2 project management activities result in an exceedingly plan-driven course of action that makes the SDP bureaucratic and slow. As noted in the value mapping – process efficiency is even lower than expected from the average plan-driven traditional methodology (see 2.3.2). Lean software development advocates the empowerment of people who are experts in the process. But, among the developers themselves, there is little understanding of why they are working in such a paperwork heavy process. They do not see the point of RFCs, mainly because the amount of paperwork halts their work progress and takes focus from developing. A more appropriate approach, and much in line with lean principles, would be to simplify the maps by elevating them to a higher abstraction level and limit unnecessary detail. Document the knowledge where necessary, but avoid using complicated tools that may require licensing or software. The use of wiki-type documentation is one way to encourage the evolution of documentation structure to best fit the knowledge of the developers, as argued in 2.3.3. Minimize static PDFs that need printing and signing by using, for example, digital signatures.

Extra Features

Interviewees indicate that the waste of creating extra features not is prevalent in the organization. This may or may not be the case, none of the empirical evidence suggests differently.

Waiting

In the development phase of the SDP, the majority of wait-states are created in handover actions between teams. At more than a few occasions, the gap has been several months or longer, this mainly because the integration team has too much work the pipeline. The problem, however, is known and efforts are taking place to allocate more resources (see 3.3.3).

Motion and Task Switching

Every time a developer needs to change context or task, both flow and time is lost in the process. The need for *task switching* and *motion* should therefore be minimized to maximize productivity. Paperwork that not adds value to the customer should be away with, and the rest should preferably be handled by teamleaders or management.

The departments of the IT function have a strictly functional setup with specialized teams (as discussed in 3.3). This cause great need of motion as teams need to be informed of other teams on a regular basis. It can be argued that the problem is less prominent for the Sundbyberg developer team, as they are located at headquarters where most of the integration, web, and database teams also coordinate their work. With regard to the Emmaboda team, however, the geographical distance is an important dimension. Interviews reveal that communication between teams (and customers) in Sundbyberg and Emmaboda has been weak at best. Virtually no cooperation between the developer teams has rendered a silo mentality among management as well as developers.

Defects

Approximately 40 percent of the Sundbyberg developer team's time is spent on maintaining current systems, which may indicate complex code and a relatively high degree of defects. The lack of quality assurance in terms of code reviews or coding standards (see 3.5.5) adds extra time when a new or inexperienced developer needs to reconfigure any part of a system. With code configuration tools such as Subversion currently being installed, the situation may improve. There is still a dire need, however, to setup routines in order to insure standardization, organization and quality of the code produced in the SDP.

Management Activities

A large project and task tracking system is used. It is, however, not used by all departments (see 3.5.2). Many projects are outdated or irrelevant, but are still maintained by the tracking system. The tracking system includes much metadata and thus requires much effort to maintain. The question is to what extent the tracking system actually brings value. The need of a tracking system increases as the lead time of tasks and project increase. A large and complex tracking system is a result of waste in the process, and also a large waste in its own right.

4.2 The Current State of Agile

The previous sections identified waste and discussed waste in, first, the entire SDP, and second, in the existence of seven common wastes according to the lean software development framework. The following section uses some of those insights, but instead focuses on the teams and their methodology and communication. Agile methodologies

have some impact, but depending on purpose, customers and technical expertise (see table 3), success of their implementation has varied from low to moderate.

Table 3. Comparison between teams in terms of purpose, customer(s), methodology and visual aids. (Source: Author)

	Sundbyberg developers	**Emmaboda developers**	**Integration**	**Web**	**Database administrators**
Purpose	Develop web and client-server solutions using Vb6, dot-net and Fortran	Develop mainframe solutions using Cobol. Develop web solutions using dot-net	Create interfaces between systems (SOA)	Enable/adjust web server in production environment	Enable/adjust application database in production environment
Customer(s)	R&D, Sales	Operations, Finance, HR	Architecture (IT dept.)	Architecture (IT dept.)	Architecture (IT dept.)
Methodology	Ad-hoc. Has a written target to implement agile methods Scrum and Kanban under 2010	Scrum to some extent. Are moving over to ultimately use Kanban only	Kanban. Purpose is to maintain focus on a few tasks at the time	Ad-hoc	Ad-hoc
Visual aids	None	Physical Scrum/Kanban board in room	Digital Kanban-board	Issue tracking system, to some extent	Issue tracking system, to some extent

Methodologies

The relationship between the Emmaboda and Sundbyberg teams has a geographical as well as a cultural dimension. The closeness to the factory seems to have made the Emmaboda developer team more open to lean practices. The opposite is also valid; i.e. the Sundbyberg developer team, being closer to head management at the headquarters, is more influenced by the parent company.

The Emmaboda team implemented Scrum, pretty much to the letter, some two years ago. But the positive effects, as 3.5 shows, are not obvious, and the implementation has been semi-functional at best. The visualization has been appreciated by managers and teamleaders alike, but the iterative approach has not worked smoothly (see 3.3.1). The deep vertical expertise among some of the developers has made it difficult or simply unmanageable to share work in the sprints.

Large RFCs cause problems. They effectively render the Scrum burndown chart useless, and are seldom completed in time with the sprint. Instead such RFCs are often returned to the backlog again, leaving the possibility of other RFCs becoming prioritized higher in the next sprint. The main reason behind this seems to be the difficulty to resize and split RFCs into work items that are small enough to complete in one sprint (see 3.5.3).

The Emmaboda team has an ambition to constantly improve and to which can be seen by their adaptation of, first, Scrum and later on, Kanban. Not at all times successfully, but with proper feedback loops they should be able to become increasingly so. The team's effort to change methodology to Kanban, however, will most likely not render any radical improvement over the current situation. The problem of coordination and

communication with other teams will still remain. What Kanban will offer though, is the opportunity to visualize wait states in the current process. With concrete stop signs (or other symbols that the team may prefer) on the board it should be much easier to discuss new ways of working with management.

At Sundbyberg, some of the same effort to improve exists, but success has proven even harder to achieve. With the developer team this is due to the strong dependency between developers and products; each developer has a large vertical expertise in a small range of systems. The integration team, on the other hand, has the horizontal expertise needed to share work items, but instead they encounter problem in the coordination with other teams. Their methodology includes a digital Kanban board to visualize workflow, both for themselves and for other teams requiring their services. But, due to lack of coordination, the board is riddled with stop signs (work items waiting for external input). They try to remedy this by adjusting the work-in-progress constraint. This defeats Kanban's main purpose (see 2.2.2) of achieving a constant and rapid flow of tasks. In conclusion, neither Scrum nor Kanban works properly in the current W&WW organization; they both encounter problem in relation to organization.

4.3 The Conflict between ITIL, Regulation and Agile

One explanation why ITIL and agile methodologies do not integrate well is found in the complexity of processes (see 2.1.1). In traditional manufacturing-inspired SDPs based on prescriptive waterfall methodologies, change is viewed as an exception, an *anomaly*. In such processes the management plans for change, trying to predict all possible risks that inflict a particular workflow. This is difficult and even, in many scenarios, practically impossible. Agile teams, conversely, are by definition based on the assumption that changes are typical events in an SDP, and nor can nor should be avoided. It's a paradox: By accepting risk and adapt to it, development should become more predictable, not the opposite.

RFCs can be described as having a small-sized traditional waterfall design. Requirements for the change are documented up front and evaluated by analysts. Stakeholders approve changes before the RFCs are even built and tested, assuming that requirements, scope, risk, and design can be formally documented ahead of making a code change. The ITIL change management process also requires extensive documentation and paperwork. The process is designed as much, if not more, to capture document change as it is to take effective decisions. Change request forms are signed, printed and kept with dates and identifiers so that the approver remains known. The teams have no mandate to drive their own changes, instead approval must come at several different stages, adding more and more wait states to the process.

In an agile project, one of the principles is to compress the team and reduce hierarchies so that people effectively can communicate (see 2.1.4). At W&WW, ITIL change management processes demands specialized knowledge, special authority of approval and a hierarchical command and control structure. Project managers or developers may be the writers of RFCs, but they must still get approval from other stakeholders who will bless the change. This inevitably slows down the process.

4.4 Suggestions For An Agile SDP

The previous section analyzes how agile and lean principles perform at W&WW, and identifies a number of obstacles impairing the agile effort. In this section the most apparent opportunities to improve are presented and discussed.

4.4.1 Application portfolio and Single-point-of-failure

The first need of the organization is to evaluate the number of supported applications. There is a large risk in having a single developer with deep vertical expertise in one or a few system. This single-point-of-failure scenario is the most critical problem in the W&WW organization, and also a bottleneck in improving the process. The problem can and should be approached from several angles at the same time:

- Review the IT portfolio: What applications are actually being used? Measure the actual use by logging access to its network path or similar. Talk to the users. How and why do they still use the application?
- What applications can be phased out over time? Setup a plan to discontinue program that has less than a given amount of uses each month
- Well-used applications that are difficult to maintain: Either (1) make sure that at least two developers have the competence to maintain the application or (2) refactor the application to the standard platform so that other developers may understand the code as well
- Implement coding standards and code reviews. As we saw in 2.3.2, good code requires less documentation, since the code itself acts as documentation. Code reviews by a second developer might seem like a waste, but will add value in the long run, simply by keeping quality high

As the risk of a single-point-of-failure scenario is reduced and the number of applications are minimized, team work within the organization should become increasingly effective.

4.4.2 Team Setup and Communication

The isolation between the Emmaboda department and the Sundbyberg department is strong. Teamleaders put their own team at first hand, although this is not the most effective way seen from a holistic lean perspective. Today, the teams are split up on a functional basis. The reasons behind the organizational change creating these functional silos are only partly revealed, but it is reasonable to think that these changes came after the acquisition of W&WW by the ITT Corporation. The larger an enterprise becomes, the more difficult it becomes to organize effectively. Today's change processes based in manufacturing and IT operational thinking, where all changes must be approved to maintain a high level of service availability in the IT systems, is, inherently ineffective for software development according to lean and agile principles.

An agile solution to improve communication is the c*ross-functional, self-organizing team* (see 2.1.4). Cross-functional team setups at W&WW are rare, but have been used at least on a few occasions (see 3.3.2). During these occasions they performed well, and at least one manager in Emmaboda expressed her satisfaction with that kind of setup. Generally, however, the attitudes towards cross-functional teams can best be described as passive, they seem to remain an exception. A major step would be to re-evaluate the change from functional to cross-functional team compositions. Cross-functional teams,

with responsibility and control of their own development process, will have much less need of task switching, team coordination and communication, and will also – maybe most importantly – take greater pride in their work.

4.4.3 A Cross-Functional Team

A change to cross-functional teams would require managers to, rather than to staff roles via separate teams (see 3.3.2), create *teams composed of analysts, architects* (web and database administrators) and *developers.* Two teams are proposed in figure 18, one with base in Emmaboda, and one with base in Sundbyberg.

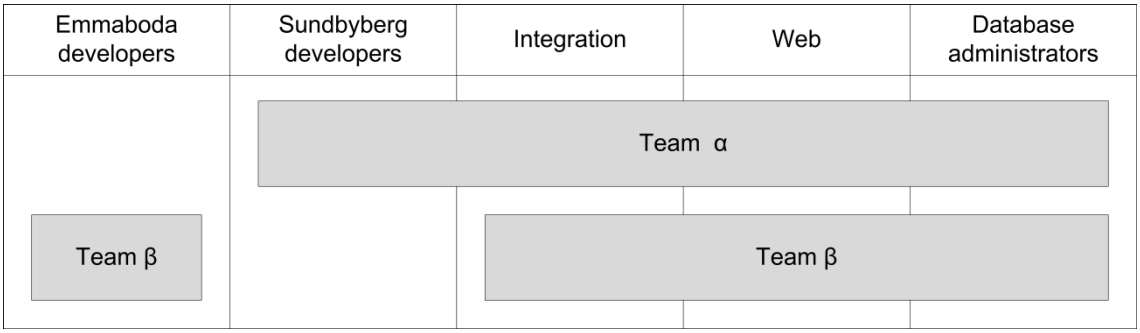


Figure 18. Two cross-functional teams with their own dedicated resources for development and architecture. Team alpha with Emmaboda lead, and team beta with Sundbyberg lead. (Source: Author)

Cross-functional teams are very effective in order to reduce the waste in handovers and communication. But to implement cross-functional teams, W&WW must find ways to integrate disparate teams efficiently. Although the geographical distance might instigate problems, these can be minimized by communication. Teams could begin each project or quarter with one initial team startup meeting (as proposed in 2.2.1) where the whole team gathers. Daily standup meetings can be held over phone or video connection. The focus of the cross-functional team would be proper information-rich, and motion-less communication rather than written documents. A natural priority order in communication would be face-to-face, telephone or voice chat, voice mail, email and last, documents (see 2.3.3). Measurable metrics would initially be lead-time and budget.

As a visual aid, a digital Kanban board could simply but efficiently describe the whole process and what is going on in the teams. Since most competence and authority that is required exist in the team, less bottlenecks (or stop signs) should clog the flow. There is still, however, external dependencies to the customer responsible team that may be more difficult to over bridge. W&WW has, however, because all stakeholders of the SDP are in-house, the advantage of being able to restructure in order to facilitate these changes.

4.5 Discussion: How to Keep Improving

So far in the analysis chapter, focus has been on how to reduce waste by thinking and acting according agile and lean principles. In this last section, I discuss how to improve from a general perspective and to what extent agile should and could be a part of such an effort.

The first thing I noticed when starting my work at W&WW was the lack of relevant key performance indicators (KPIs). There was no way to tell whether the organization was effective or not in terms of productivity. There was time-sheet numbers on the amount

of work spent in maintenance and on different projects, but that did not really speak much about how the process itself performed. If the output of the SDP was measured, improvement could more easily be identified. The recent installation of a single software versioning and revision control system should make this easier. A basic KPI for productivity can be defined, perhaps using story points or other metrics as the unit of measurement. A second tool, the value stream map, could act as a powerful complement to the KPIs to continuously improve the lead time of the process.

The second thing I noticed when starting up my work at W&WW was the uncertainty of the organizational structure. Numerous changes had made the internal communication unclear. There was also a firm ambition of integrating the W&WW value center with other parts of the ITT Corporation and the Fluid technology business unit. These changes were made in the name of economies of scale, but they also brought something else with them – a large and bureaucratic organization. As the organization grows, improvement programs will inevitable become larger and local solutions less flexible. ITIL and PULS2 set a framework that may be difficult to change.

Best practice frameworks and improvement programs, as is also argued by Poppendieck et al (2009), has a tendency to lock the organization in a particular way of thinking. This is precarious for W&WW. With its manufacturing part being strong, W&WW should keep in mind that lean is a constant search for improvement by those who know the process best – developers and team leaders.

The SDP of today is definitely of a plan-driven nature, making it slow but probably relatively safe in terms of ensuring the availability of the IT infrastructure. Traditionally, the watering and dewatering business have not been overly competitive, at least not in terms of IT tools. But with an increasing focus on staying ahead in the configurator tool area, the need to stay alert in software development may also increase. One can safely assume that IT will emerge as, if not the most important, at least one of the most important tools in staying ahead in the water handling business. Development will also remain difficult to outsource, as some of the applications are very knowledge intensive, with calculation models that requires very specialized skills to properly construct. The very nature of Scrum, with a strong focus on adapting to changing customer requirements and iterative releases, may not be imperative to implement right now – but who knows about tomorrow?

5. Results

The study aimed to reveal waste in a typical software development process of a large manufacturing-oriented organization, and identify obstacles in applying a lean software development framework to remove such waste.

Lean software development is in conflict with many traditional values of manufacturing organizations. Although lean may be prevalent in other parts of the organization, this does not necessarily include the IT function. IT still has a hard time comprehending the benefits of concepts such as flow, waste and value. At W&WW, a large project tracking system and long lead times indicate that waste is a valid concern. The most notable waste is wait time for change approval meetings. A second prominent waste is the abundance of handshake actions in granting approvals at different stages of the process. At least a few of these handshakes seems to be approvals that always approve and, as such, they are also very wasteful.

The first obstacle in adopting Lean software development is *deep vertical expertise*. A few developers are experts in a narrow set of applications. With some systems, there's only one developer who knows how to maintain the product. This makes it impossible to work as a team, which is an imperative principle of lean. A second obstacle is how the teams are arranged *organizationally*. They have a functional setup over three departments and three managers, which to some extent create a silo mentality, rendering cooperation difficult. A third obstacle is how the teams are arranged *geographically*. Split over two locations, manufacturing and headquarters, different customers, objectives and plain unfamiliarity has reduced the will and opportunity to communicate and coordinate. A fourth obstacle is the inherent conflict between the prescriptive activities of ITIL, optimized for IT operational services, and the adaptability of agile methodologies, optimized for rapid change and empirical decisions. ITIL fulfills a sometimes uncalled for need to get all changes approved through several layers of management.

5.1 Further Studies

As work with this thesis evolved, many interesting aspects of agile came into light. One is the conflict between plan-driven and agile methodologies. How would agile perform in a highly regulative business area such as defense or medical appliances? Does agile impose a less traceable and less secure development process? Another interesting aspect would be to survey the views of developers and managers on the benefits and downsides of adopting Scrum and Kanban. How do they differ? How does the role of a manager change in an agile approach?

References

Books

Beck, K. (2004), *Extreme Programming Explained: Embrace Change (2nd Edition)*, Addison-Wesley Professional

Bjørnvig, G. & Coplien, J. (2010), *Lean Architecture: for Agile Software Development*, Hoboken: Wiley

Cohn, M. (2010), *Succeeding with Agile: Software Development Using Scrum*. Boston: Addison-Wesley Professional

McConnell, S. (2004), *Code Complete: A Practical Handbook of Software Construction*, Microsoft Press, San Francisco

McConnell, S. (1996), *Rapid Development: Taming Wild Software Schedules*, Microsoft Press, San Francisco

Poppendieck, M., & Poppendieck, T. (2009), *Leading Lean Software Development: Results Are not the Point*, Crawfordsville: Addison-Wesley Professional

Poppendieck, M., & Poppendieck, T. (2006), *Implementing Lean Software Development: From Concept to Cash*, Addison-Wesley Professional

Poppendieck, M., & Poppendieck, T. (2003), *Lean Software Development: An Agile Toolkit*, Crawfordsville: Addison-Wesley Professional

Royce, W. (1970), *Managing the development of large software systems: concepts and techniques*, Proceedings of the 9th international conference on Software Engineering

Schwaber, K. (2004), *Agile Project Management with Scrum*, Microsoft Press, San Francisco

Reports

Boehm, B. & Turner, R. (2004), *Balancing Agility and Discipline: A Guide for the Perplexed*, Boston, MA: Addison-Wesley

Brooks, F. (1995), *No Silver Bullet' Refired*, Addison-Wesley

Brooks, F. (1986), *No Silver Bullet — Essence and Accidents of Software Engineering*, Proceedings of the IFIP Tenth World Computing Conference: 1069–1076

Dubakov, M. & Stevens, P. (2008), *Agile Tools. The Good, the Bad and the Ugly*, Target Process Inc. White Paper, New York, USA. Available online: www.cs.odu.edu/~cs350m/Docs/agiletools.pdf

Kniberg, H. (2007), *Scrum and XP from the Trenches*, 4Media Inc. InfoQ.com. Available online: http://www.infoq.com/minibooks/Scrum-xp-from-the-trenches

Kniberg, H. & Skarin, M. (2010), *Kanban and Scrum: Making the most of both*, 4Media Inc. InfoQ.com. Available online: www.infoq.com/minibooks/Kanban-Scrum-minibook

Knight, J. et al (1997), *Why Are Formal Methods Not Used More Widely?*, Department of Computer Science, University of Virginia, Charlottesville, VA 22903

Mah, M. (2008), *How Agile Projects Measure Up and What This Means To You*, Cutter Consortium Vol. 9, No. 9

Ohno, T. (1988), *Toyota Production System*, Productivity Press

Shingo, S. (1989), *A Study of the Toyota Production System*, Productivity Press

Waterhouse, P. (2008), *Improving IT Economics: Thinking 'Lean'*, CA White Paper, New York, USA. Available online: www.ca.com/files/WhitePapers/improving-it-economics-wp.pdf

Weinberg, G. (2003), *Iterative and Incremental Development: A Brief History*, Published by the IEEE Computer Society. Available online: http://www.highproductivity.org/r6047.pdf

Internet

Agile Alliance official site (2010), *The Alliance: Mission and operations*, www.agilealliance.org/the-alliance/ (2010-01-03)

Ambler, S. (2007), *2007 IT Project Success Rates Survey Results*, www.ambysoft.com/surveys/success2007.html (2011-01-05)

Appelo, J. (2008), *Simple vs. Complicated vs. Complex vs. Chaotic*, www.noop.nl/2008/08/simple-vs-complicated-vs-complex-vs-chaotic.html (2011-01-09)

ITIL Official Site (2010), *What is ITIL?*, www.itil-officialsite.com/AboutITIL/WhatisITIL.asp (2010-12-13)

ITT Corporation official site (2010), *About*, www.itt.com/about/ (2011-01-09)

ITT Fluid Technology (2010), *Profile*, ittfluidbusiness.com/profile.htm (2011-01-10)

Johnson, J. (2002), *Standish Group: Third International Conference on Extreme Programming (XP2002)*, luuduong.com/blog/archive/2009/03/04/applying-the-quot8020-rulequot-with-the-standish-groups-software-usage.aspx (2010-01-12)

Lundgren, M. (2010), *Introduktion till lättrörlig utveckling: med Lean och Scrum,* www.scribd.com/doc/27454567/Introduktion-till-lattrorlig-utveckling (2011-01-09)

Manifesto for Agile Software Development (2001), *Principles behind the Agile Manifesto*, agilemanifesto.org/principles.html (2011-01-09)

Nielsen, D. (2010). *How to Control Change Requests*, www.pmhut.com/how-to-control-change-requests (2010-12-13)

Reinertsen, D. (2007), *A Lean Rethink*, www.reinertsen.co.uk/LeanRethink.html (2011-01-09)

Richmond Systems (2010), *Glossary of terms*, www.helpdesksoftware-richmond.co.uk/glossary/glossary-I.htm (2011-01-09)

Schwaber, K. (2010), *Ken Schwaber's Blog: Telling It Like It Is,* kenschwaber.wordpress.com (2010-12-16)

Scrum methodology (2009), *Defined System Development Methodologies*, www.Scrummethodology.org/development-methodologies.html (2011-01-09)

Shore, J. (2004), *Task Switching*, jamesshore.com/Articles/Business/Software%20Profitability%20Newsletter/Task%20Switching.html (2010-12-13)

Stine, M. (2010), *Waste #6: Task Switching*, agile.dzone.com/articles/waste-6-task-switching (2010-01-03)

Studbler, R (2008), *ITIL and Software Development*, www.gl-spin.org/2008-11.ppt (2010-12-13)

Software Benchmarking Organization (2010), *About SBO*, www.sw-benchmarking.org (2011-01-09)

Weisert, C. (2003), *There's no such thing as the Waterfall Approach!*, www.idinews.com/waterfall.html (2010-12-13)

W&WW official site (2010), *About*, www.ittwww.com/3256264.asp (2011-01-09)

W&WW Intranet (2010a), *About*, group.ittwww.com (2011-01-09)

W&WW Intranet (2010b), *Global sourcing*, group.ittwww.com (2011-01-09)

W&WW Intranet (2010c), *Collaboration space*, group.ittwww.com (2011-01-09)

W&WW Intranet (2010d), *PAR*, group.ittwww.com (2011-01-09)

W&WW intranet (2010e), *control document 03768*, group.ittwww.com (2011-01-09)

W&WW Intranet (2009), *PULS2 control document*, group.ittwww.com (2011-01-09)

Oral Sources

Teamleader A (2010), interview, Sundbyberg, 2010-10-12

Teamleader B (2010), interview, Sundbyberg, 2010-10-15

Teamleader C and D (2010), interview, Sundbyberg, 2010-10-20

Teamleader E (2010), interview, Sundbyberg, 2010-11-24

Teamleader F (2010), interview, Emmaboda, 2010-10-26

Scrum Master (2010), interview, Emmaboda, 2010-10-26

Manager (2010), telephone interview, 2010-11-18

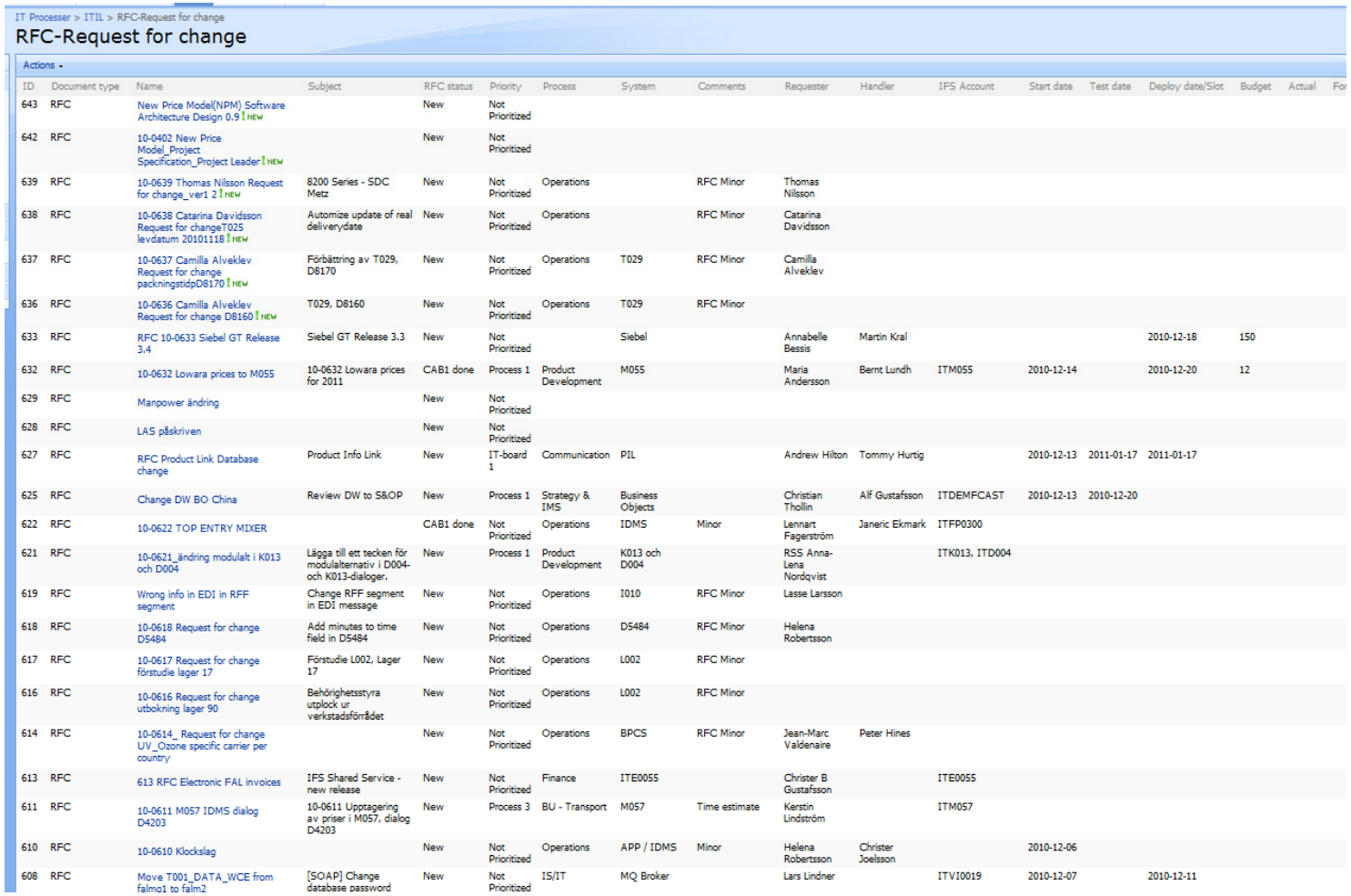
Developer A (2010), interview, Emmaboda, 2010-10-26

Developer B (2010), interview, Sundbyberg, 2010-11-11

Purchaser (2010), interview, Emmaboda, 2010-10-26

Appendix I

Project Tracking System



IT Processer > ITIL > RFC-Request for change

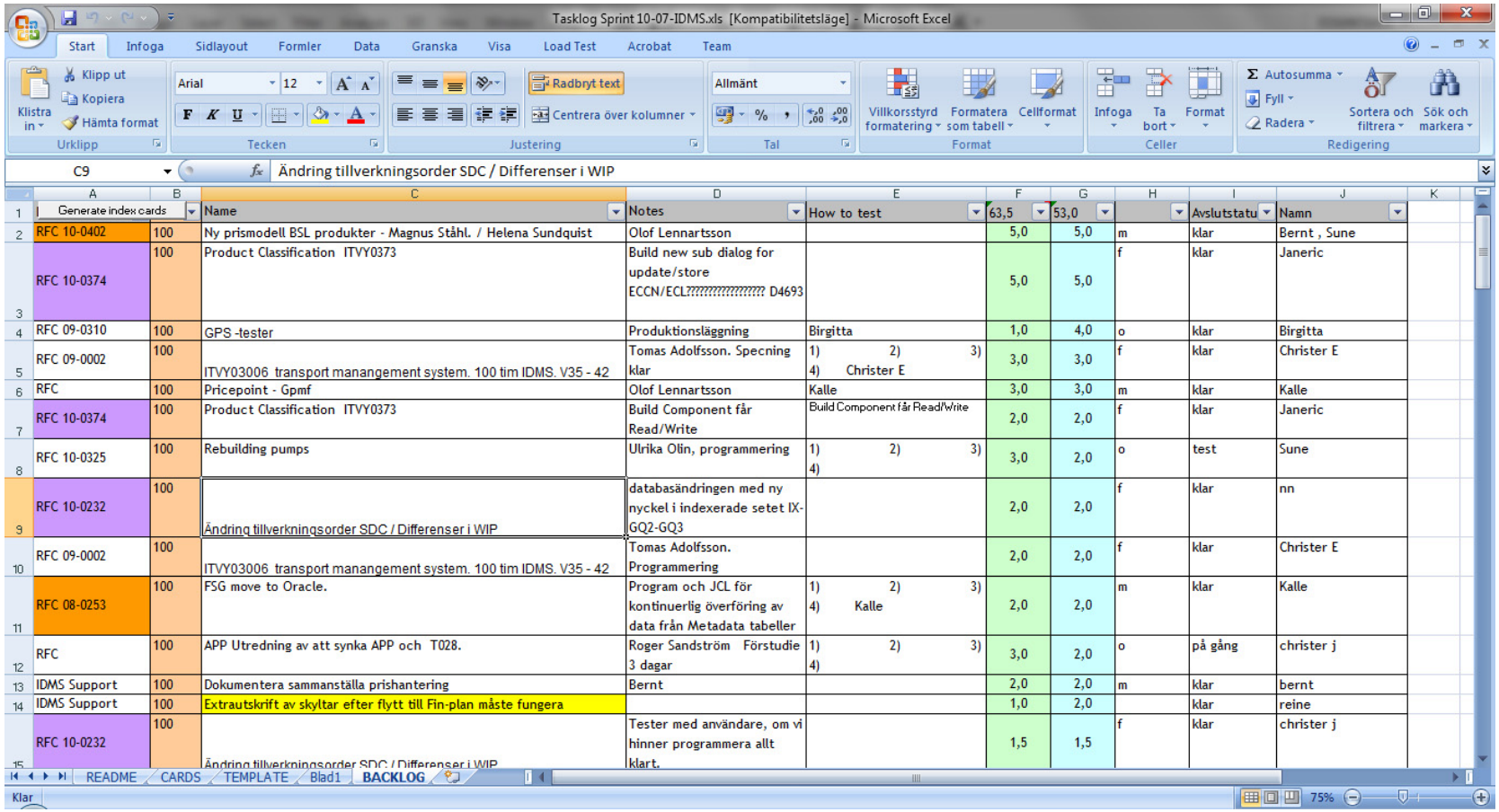
RFC-Request for change

Actions

ID	Document type	Name	Subject	RFC status	Priority	Process	System	Comments	Requester	Handler	IFS Account	Start date	Test date	Deploy date/Slot	Budget	Actual	For
643	RFC	New Price Model(NPM) Software Architecture Design 0.9 NEW		New	Not Prioritized												
642	RFC	10-0402 New Price Model_Project Specification_Project Leader NEW		New	Not Prioritized												
639	RFC	10-0639 Thomas Nilsson Request for change_ver1 2 NEW	8200 Series - SDC Metz	New	Not Prioritized	Operations		RFC Minor	Thomas Nilsson								
638	RFC	10-0638 Catarina Davidsson Request for changeT025 levdatum 20101118 NEW	Automize update of real deliverydate	New	Not Prioritized	Operations		RFC Minor	Catarina Davidsson								
637	RFC	10-0637 Camilla Alveklev Request for change packningstidpD8170 NEW	Förbättring av T029, D8170	New	Not Prioritized	Operations	T029	RFC Minor	Camilla Alveklev								
636	RFC	10-0636 Camilla Alveklev Request for change D8160 NEW	T029, D8160	New	Not Prioritized	Operations	T029	RFC Minor									
633	RFC	RFC 10-0633 Siebel GT Release 3.4	Siebel GT Release 3.3	New	Not Prioritized		Siebel		Annabelle Bessis	Martin Kral				2010-12-18	150		
632	RFC	10-0632 Lowara prices to M055	10-0632 Lowara prices for 2011	CAB1 done	Process 1	Product Development	M055		Maria Andersson	Bernt Lundh	ITM055	2010-12-14		2010-12-20	12		
629	RFC	Manpower ändring		New	Not Prioritized												
628	RFC	LAS påskriven		New	Not Prioritized												
627	RFC	RFC Product Link Database change	Product Info Link	New	IT-board 1	Communication	PIL		Andrew Hilton	Tommy Hurtig		2010-12-13	2011-01-17	2011-01-17			
625	RFC	Change DW BO China	Review DW to S&OP	New	Process 1	Strategy & IMS	Business Objects		Christian Thollin	Alf Gustafsson	ITDEMFCAST	2010-12-13	2010-12-20				
622	RFC	10-0622 TOP ENTRY MIXER		CAB1 done	Not Prioritized	Operations	IDMS	Minor	Lennart Fagerström	Janeric Ekmark	ITFP0300						
621	RFC	10-0621_ändring modulalt i K013 och D004	Lägga till ett tecken för modulalternativ i D004- och K013-dialoger.	New	Process 1	Product Development	K013 och D004		RSS Anna-Lena Nordqvist		ITK013, ITD004						
619	RFC	Wrong info in EDI in RFF segment	Change RFF segment in EDI message	New	Not Prioritized	Operations	I010	RFC Minor	Lasse Larsson								
618	RFC	10-0618 Request for change D5484	Add minutes to time field in D5484	New	Not Prioritized	Operations	D5484	RFC Minor	Helena Robertsson								
617	RFC	10-0617 Request for change förstudie lager 17	Förstudie L002, Lager 17	New	Not Prioritized	Operations	L002	RFC Minor									
616	RFC	10-0616 Request for change utbokning lager 90	Behörighetsstyra utplock ur verkstadsförrådet	New	Not Prioritized	Operations	L002	RFC Minor									
614	RFC	10-0614_ Request for change UV_Ozone specific carrier per country		New	Not Prioritized	Operations	BPCS	RFC Minor	Jean-Marc Valdenaire	Peter Hines							
613	RFC	613 RFC Electronic FAL invoices	IFS Shared Service - new release	New	Not Prioritized	Finance	ITE0055		Christer B Gustafsson		ITE0055						
611	RFC	10-0611 M057 IDMS dialog D4203	10-0611 Upptagering av priser i M057, dialog D4203	New	Process 3	BU - Transport	M057	Time estimate	Kerstin Lindström		ITM057						
610	RFC	10-0610 Klockslag		New	Not Prioritized	Operations	APP / IDMS	Minor	Helena Robertsson	Christer Joelsson		2010-12-06					
608	RFC	Move T001_DATA_WCE from falmo1 to falm2	[SOAP] Change database password	New	Not Prioritized	IS/IT	MQ Broker		Lars Lindner		ITVI0019	2010-12-07		2010-12-11			

Appendix II

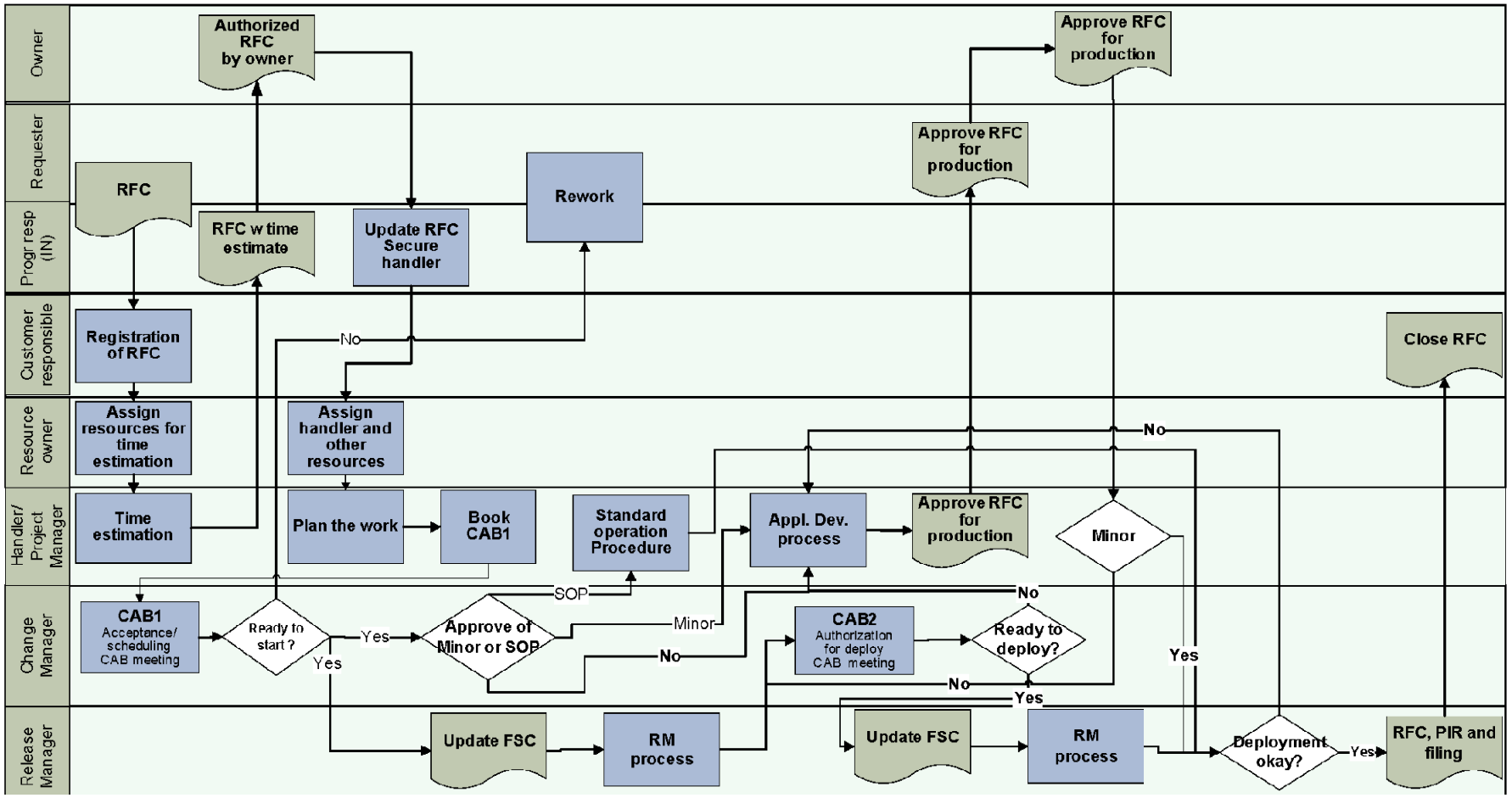
Emmaboda team's RFC Tracking Spreadsheet



	A	B	C	D	E	F	G	H	I	J	K
1	Generate index cards		Name	Notes	How to test	63,5	53,0		Avslutstatu	Namn	
2	RFC 10-0402	100	Ny prismodell BSL produkter - Magnus Ståhl. / Helena Sundquist	Olof Lennartsson		5,0	5,0	m	klar	Bernt , Sune	
3	RFC 10-0374	100	Product Classification ITVY0373	Build new sub dialog for update/store ECCN/ECL??????????????? D4693		5,0	5,0	f	klar	Janeric	
4	RFC 09-0310	100	GPS -tester	Produktionsläggning	Birgitta	1,0	4,0	o	klar	Birgitta	
5	RFC 09-0002	100	ITVY03006 transport manangement system. 100 tim IDMS. V35 - 42	Tomas Adolfsson. Specning klar	1) 2) 3) 4) Christer E	3,0	3,0	f	klar	Christer E	
6	RFC	100	Pricepoint - Gpmf	Olof Lennartsson	Kalle	3,0	3,0	m	klar	Kalle	
7	RFC 10-0374	100	Product Classification ITVY0373	Build Component får Read/Write	Build Component får Read/Write	2,0	2,0	f	klar	Janeric	
8	RFC 10-0325	100	Rebuilding pumps	Ulrika Olin, programmering	1) 2) 3) 4)	3,0	2,0	o	test	Sune	
9	RFC 10-0232	100	Ändring tillverkningsorder SDC / Differenser i WIP	databasändringen med ny nyckel i indexerade setet IX-GQ2-GQ3		2,0	2,0	f	klar	nn	
10	RFC 09-0002	100	ITVY03006 transport manangement system. 100 tim IDMS. V35 - 42	Tomas Adolfsson. Programmering		2,0	2,0	f	klar	Christer E	
11	RFC 08-0253	100	FSG move to Oracle.	Program och JCL för kontinuerlig överföring av data från Metadata tabeller	1) 2) 3) 4) Kalle	2,0	2,0	m	klar	Kalle	
12	RFC	100	APP Utredning av att synka APP och T028.	Roger Sandström Förstudie 3 dagar	1) 2) 3) 4)	3,0	2,0	o	på gång	christer j	
13	IDMS Support	100	Dokumentera sammanställa prishantering	Bernt		2,0	2,0	m	klar	bernt	
14	IDMS Support	100	Extrautskrift av skyltar efter flytt till Fin-plan måste fungera			1,0	2,0		klar	reine	
15	RFC 10-0232	100	Ändring tillverkningsorder SDC / Differenser i WIP	Tester med användare, om vi hinner programmera allt klart.		1,5	1,5	f	klar	christer j	

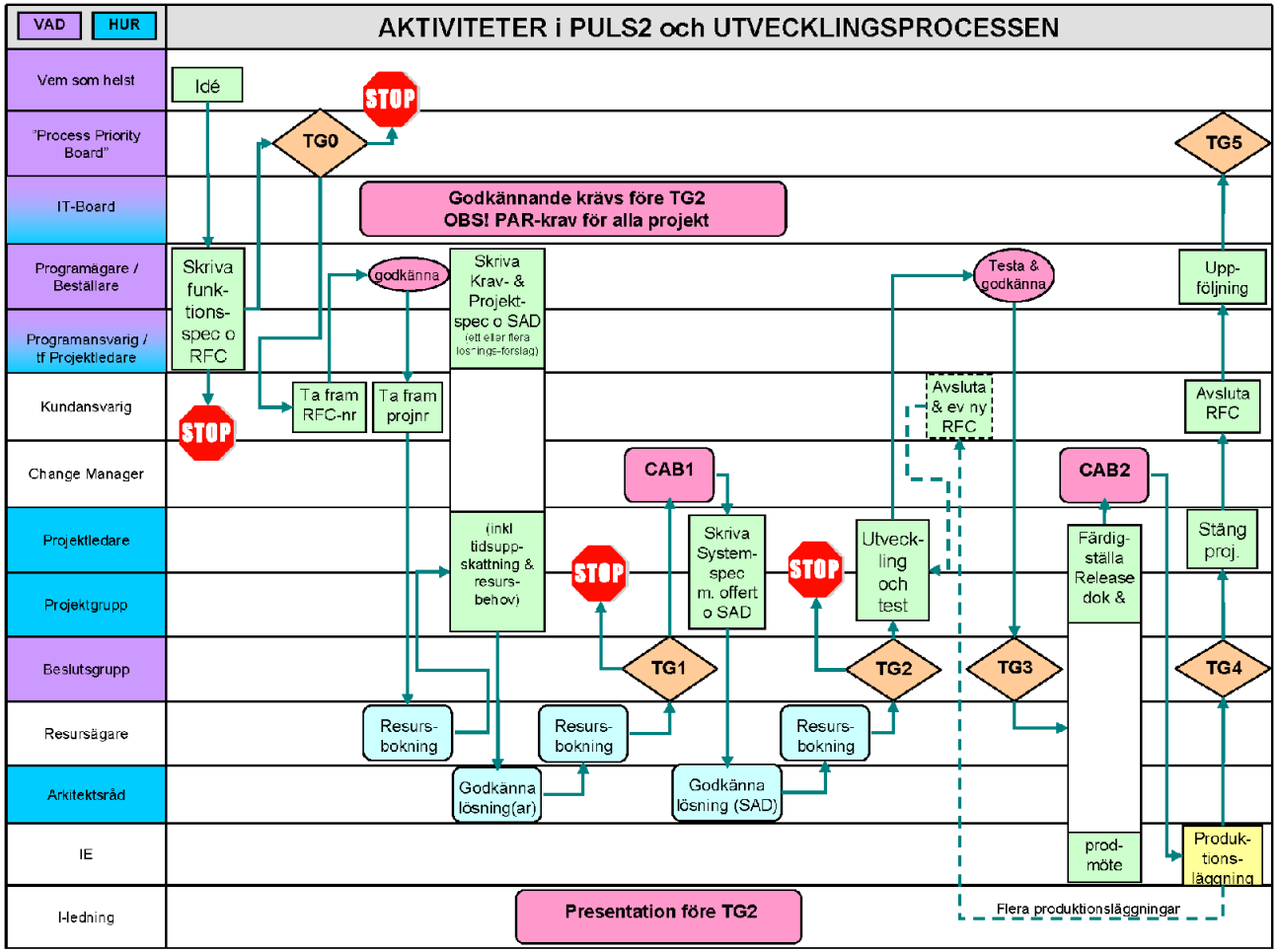
Appendix III

The Change Management Process



Appendix IV

The Change Management Process with PULS2 (in Swedish)



Appendix V
Interview Topics (in Swedish)

Overview with Supervisor

Allmänt om IT-verksamheten

- Antal anställda
- Avdelningar
- Geografisk utbredning
- Organisationsschema

Allmänt om systemutveckling

- Vad utvecklas?
- Vem används programmen av?
- Vem äger förvaltningen av programmen?
- Hur mycket är nyutveckling kontra vidareutveckling?
- Hur sker vidareutveckling av programmen?
- Hur många utvecklar?
- Vem är kund?

Teamen

- Hur är teamen uppdelade? Varför?
- Hur är teamen placerade geografiskt?
- Hur ser arbetssättet ut för ett team i termer av antal projekt, underhåll/vidareutveckling av existerande system…

Projekt

- Vad menar ni med ett projekt?
- Hur ser ett typiskt projekt ut?
- Arbetar ni alltid i projektbaserat arbete?
- Hur omfattande är projekt i termer av utvecklingstid?
- Vilka är involverade?
- Kundens/användarens roll i projektet?

Produktivitet och benchmarking

- Hur upplever ni att verksamheten fungerar?
- Hur mäter ni mjukvarutvecklingens prestation?
- Produktivitet?
- Jämför ni er med andra företag?

Leanfilosofi i företaget

- …Organisationen som helhet
- … I it-verksamheten
- Hur ser man på framtiden för lean?
-

Agil utveckling i företaget

- …Organisationen som helhet
- … I it-verksamheten
- Hur ser man på framtiden för agil utveckling?
- Finns Scrum i organisationen idag?

Diskussion kring relevanta personer att prata med

- Utvecklare
- Projektledare
- Ägare
- IT-chef
- Representanter från VBLSS

Teamleaders (in Swedish)

Organisation

- Vad är teamets syfte?
- Hur många utvecklare är ni och hur är rollfördelningen?
- Hur mycket av teamets verksamhet är nyutveckling kontra förvaltning/underhåll av existerande system?
- Är all nyutveckling i projektform?
- Hur många projekt är de aktiva i?
- Vem är kunden för era projekt?
- Vilka krav ställer kunden på er?
- Hur ger kunden feedback till er på hur systemen fungerar?

Utveckling

- Vilken typ av lösningar utvecklas?
- Hur väljer ni systemarkitektur för era lösningar?
- Vilka riktlinjer finns för hur kod ska skrivas?
- Ett typiskt projekt
- Följer era utvecklingsprojekt Puls2-mallen?
- Hur påbörjas projektet?
- Omfattning, i termer av personal och tid?
- Om resurser saknas, vad gör ni?

- Sammansättning i termer av konsulter och anställda?
- När anses projektet var slut?
- Vad är kundens roll i utvecklingsprocessen?
- Hur är kontakten mellan er och systems faktiska användare?
- På vilket sätt arbetar ni med andra team?
- Hur stort är beroendet av andra team?
- Hur påverkas ni av geografisk spridning?
- Hur drar ni erfranheter från ett avslutat projekt?

Utvecklingsmetodik

- Vilka problem har ni sett med dagens arbetssätt?
- Hur ser ni på användning av Scrum respektive Kanban?
- Varför är ni intresserade av just dessa metoder?
- Finns det någon plan eller direktiv uppifrån för att ändra arbetssätt?
- I vilket stadium befinner ni er i införandet av sådan metodik?
- Hur ser ni på koncept som ...
    - Product owner
    - Standup meetings
    - Tavlor

Emmaboda Developer Team

Scrum

- Varför har ni övergått till Scrum? Varför denna metod?
- Finns det någon plan och/eller direktiv uppifrån för att ändra arbetssätt?
- I vilket stadium befinner ni er i införandet av Scrum?
- Hur går ni till väga för att ändra arbetssätt?
- Berätta om er Scrum-metodik. Hur, konkret, använder ni koncept som
    - Standup meetings
    - Sprint planning meetings
    - Sprints
    - Backlog
    - Sprint backlog
    - Product owner
    - ScrumMaster
- Hur ser ni på framtiden för Scrum i Emmaboda?

ScrumMaster

- Berätta om din roll i teamet.
- Vilken bakgrund har du?
- Hur länge har du varit ScrumMaster?
- Varför blev du ScrumMaster?

Customer and User

Användare

- Vilken avdelning tillhör du?
- Vad är syftet med din avdelning?
- Vilket system använder du?
- Hur länge har du arbetat med det här systemet?
- Vad gör du då du får ett problem med ditt system?
- Vad gör du då saknar eller har förslag på en ny funktion i systemet?
- Ger du annan typ av feedback från/till produktansvarig eller utvecklingsteamet?
- Upplever du att utvecklingstemat/produktansvarig är lyhörda för dina frågor?

Management

ITIL och IT-processer

- Vilka delar av ITIL är implementerade?
- hur använder ni er av dessa delar? Följer projektledare processkartorna?
- Hur påverkas IPD av ITIL i den dagliga verksamheten?
- Gå igenom Change Management-processen.
    - Vilka sitter i CAB-mötena?
    - Hur sätts denna grupp samman?
    - Vem/vilka innehar de olika rollerna och varför?
- Hur definieras application development-processen? Hur och av vem togs processen fram?

Emmaboda developer team - IPD

- Hur tycker du att dotnet-utvecklarnas respektive stordatorutvecklarnas arbete har förändrats sedan Scrum infördes?
    - Vad fungerar bättre nu?
    - Har något blivit sämre?
- Hur motiverade ni er övergång till Scrum?
- Har ni diskuterat hur ni kan utveckla metodiken ytterligare?

Samarbete

- Hur du på samarbetet mellan IPD, integratörer, webb och DBAs? Vad fungerar bra och vad fungerar mindre bra?
- Hur ser du på tvärfunktionella teams?
- Hur ser du på framtida samarbeten mellan IND och IPD?

Ledningsgrupp

- Hur ofta har ni ledningsgruppsmöten och vilka medverkar?
- Vilken typ av diskussioner har du med andra avdelningsansvariga?

- Hur koordinerar ni IN, IP och architecture IEs verksamheter?
- Hur mäter ni eller uppskattar verksamhetens effektivitet?
- Målen för 2010 innehöll en övergång till agila utvecklingsmetoder, utryckligen ”Scrum eller Kanban”.
    - Vad förväntar ni er av en sådan övergång?
    - Vilka är anledningar till övergå till agil utveckling?

Topics for Discussion with Teamleaders

Hur arbetar man Emmaboda

- De positiva/negativa erfarenheter som upplevts i Scrum jämförelse med tidigare arbetssätt
- Hur de löser det hela rent praktiskt, i termer av
    - Möten/planering/stand up meetings
    - Tavla
    - RFCs och prioriteringar i backloggen
    - Interationer/sprintar
    - Avstämningar och uppföljningar med kund
    - Flera utvecklingsplattformar och spridda kompetenser hos utvecklarna
    - Kodstandarder/konventioner
    - Synkronisering av sina sprintar mot IEI:s Kanban
- Hur de ser på framtiden, i termer av att utveckla sin Scrum-metod ytterligare

Jämförelse med IEI och IND

- Vilka fördelar har deras metod?
- Vilka styrkor och svagheter finns?
- Hur står den sig kontra Kanban?
- Vad kan man göra idag och vad kräver längre tid?
- Hur IPD ser på samarbetet med er.
    - Vad fungerar och vad fungerar inte.
    - Hur kan det utvecklas?