Aalto University
School of Science
Department of Computer Science and Engineering

Miika Leskinen

Towards a flow efficient ICT development process with Kanban: A Case Study

Master's Thesis
Helsinki, 21.5.2015

Supervisor: Professor Casper Lassenius
Advisor: Toni Pallaspuro, MSc (Econ.)

Aalto University
School of Science
Department of Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Miika Leskinen

Title: Towards a flow efficient ICT development process with Kanban: A Case Study

Date: May 21, 2015 **Pages:** 99+2

Professorship: Software Engineering and Business **Code:** T-76

Supervisor: Professor Casper Lassenius

Advisor: Toni Pallaspuro M.Sc (Econ.)

Kanban method has been an emerging topic in the field of software engineering after the agile software development methods, such as Scrum and XP, were introduced. Today in competitive business environments, organizations strive for efficient development processes in order to stay competitive and satisfy the business needs.

The objective of this thesis is to study the ICT development process in Finnish Transport Safety Agency Trafi and to identify the barriers, which are causing inefficiency in the process. Additionally, using literature this thesis studies how the Kanban method can increase the performance of a process.

The results confirm that the current ICT development process is not efficient and there are multiple bottlenecks in the process. The biggest bottlenecks in the case organization are internal resourcing policy, dependencies, and the deployment process.

The findings of this thesis support the adaption of the Kanban method in the case organization. Principles of the Kanban method should be adopted in the organizational level in order to remove the biggest bottlenecks from the ICT development process.

Keywords: Kanban, flow efficiency, Scrum, Software engineering, Agile

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Miika Leskinen		
Työn nimi:	Towards a flow efficient ICT development process with Kanban: A Case Study		
Päiväys:	21. toukokuuta 2015	**Sivumäärä:**	99+2
Professuuri:	Ohjelmistotuotanto ja liiketoiminta	**Koodi:**	T-76
Valvoja:	Professori Casper Lassenius		
Ohjaaja:	Ekonomi Toni Pallaspuro		

Kanban menetelmä on noussut esille ohjelmistotuotannossa ketterien ohjelmistokehitysmenetelmien, kuten Scrumin ja XP:n, yleistymisen jälkeen. Tänä päivänä organisaatiot pyrkivät saavuttamaan tehokkaan kehitysprosessin pysyäkseen kilpailukykyisinä ja täyttääkseen liiketoiminnan tarpeet.

Tämän diplomityön tavoitteena on tutkia Liikenteen turvallisuusvirasto Trafin nykyistä ICT-kehitysprosessia ja tunnistaa siinä olevat haasteet, jotka aiheuttavat tehottomuutta prosessille. Lisäksi työssä tutkitaan kirjallisuutta hyödyntäen miten Kanban menetelmä voi tehostaa kehitysprosesseja.

Työn tulokset vahvistavat käsitystä, että nykyinen ICT-kehitysprosessi ei ole optimaalinen ja että siinä on lukuisia pullonkauloja. Organisaation suurimmat pullonkaulat ovat sisäisten resurssien käyttö, riippuvuudet sekä tuotannointiprosessi.

Tämän diplomityön löydökset tukevat Kanban menetelmän käyttöön ottamista tutkittavassa organisaatiossa. Kanbanin periaatteet tulisi ottaa käyttöön kehitystiimien sisäisten prosessien lisäksi koko organisaation tasolla, jotta koko ICT-kehitysprosessin suurimmat pullonkaulat saadaan poistettua.

Asiasanat:	Kanban, virtaustehokkuus, Scrum, ohjelmistokehitys, Ketterä ohjelmistokehitys
Kieli:	Englanti

Acknowledgements

This thesis would not have been possible without help and support of many people. First of all, I want to thank my thesis supervisor for all the support and guidance during the thesis. I also want to thank my thesis advisor Toni Pallaspuro for giving me the opportunity and enabling me to study this interesting topic at Trafi.

My sincere thanks go to all the interviewees who participated in the interviews and provided me the necessary data for the thesis. My humble thanks also go to everyone else at Trafi who have helped me during the study and gave me valuable thoughts considering the topic. Especially I want to thank Matti Riihijärvi and Tuomas Jokinen for sharing their thoughts and ideas about Kanban and flow efficiency. In addition, I want to thank Mikko Vuorikoski for his contribution in the early phases of the thesis.

Helsinki, May 21, 2015

Miika Leskinen

Abbreviations and Acronyms

APMM	Adaptive Project Management Model is a process model for development work at Trafi.
BN	Business Need, e.g. a new software feature
CFD	Cumulative Flow Diagram
DLCM	Development Life Cycle Model is the high-level process model for projects at Trafi.
ICT	Information and Communication Technology
KARKKI	An acronym for Kokonaisarkkitehtuuri (Enterprise Architecture). KARKKI team is responsible for the Development Life Cycle Model.
PMO	Project Management Office
PO	Product Owner, a role defined in Scrum
WIP	Work-In-Progress

Content

1 Introduction

1.1 Motivation

Software applications have become an important part in people's lives during the past few decades since software is now included in vast amount of products (Petersen & Wohlin, 2009). The importance of software is even more critical to companies and corporations in their businesses. Traditionally, software systems were delivered to the customers as a one big release and the delivery times were long. However, today in highly competitive markets, successful software business requires faster respond to the customer needs (Dybå & Dingsøyr, 2008). Agile software development methods were introduced to address this issue among many additional issues related to software development (Ahmad, et al., 2013). In recent years, the research community has started to look toward lean software development approaches such as Kanban (Wang, et al., 2012).

Lean philosophies originate from the Japanese automobile industry in 1950s' (Japan Management Association, 1989). Within the past few decades, Lean has spread widely across industries, including in the software engineering industry. David J. Anderson publically introduced Kanban method, which is derived from Lean philosophies for software engineering professionals in the context of software engineering through a book in 2010 (Anderson, 2010). The main focus behind the Kanban method is to visualize the workflow and limit the work in progress, in order to achieve a flow efficient development process (Anderson, 2010). The dynamic nature of software development including unrealistic schedules and requirement changes are taken into account in the Kanban method (Ikonen, et al., 2011)

Current way of management in most organizations is to use the existing resources efficiently. Resource efficiency focuses on utilizing the value-adding resources, such as people, machines or information as efficiently as possible (Modig & Åhlström, 2012). Another form of efficiency is the flow efficiency, which focuses on minimizing the time from identifying a need to satisfying the need (Modig & Åhlström, 2012). The benefits of Kanban in increasing the process efficiency are clearly recognized in the manufacturing industry (Japan Management Association, 1989).

The existing research (case studies) in the field of software engineering does not provide a clear indication that Kanban method help organizations to increase flow efficiency in their software development processes. To address this issue, we first attempt to understand how the Kanban method solves the process challenges in software engineering and how Kanban has been taken into use in different organizations. Second, we seek to understand the ICT development process at Finnish Transport Safety Agency Trafi (from now on referred only as Trafi) in order to identify the current challenges that are preventing the process to work flow efficiently. Third, we gather the findings from the existing research and apply it in the case organization. Through these phases, we gain a better understanding of whether the Kanban method has the potential to improve flow efficiency in software development process.

1.2 Motivation in case organization

In the case organization (Trafi), ICT development has been challenging for long. Release cycles have been long (from 6 to 12 months), the final quality of the software systems with long delivery times in waterfall projects have been unsatisfying, original schedules and budgets have been constantly underestimated. Since fall 2012, transformation towards agile software development methods has been active. First the existing development projects were transformed towards agile software development framework called Scrum (Schwaber & Sutherland, 2013), and later each new project was started with an agile mindset. Even though Scrum has been introduced in the case organization, the current pace of the software deliveries is still too slow to fulfill the business needs. Regulations, laws, technologies, and even the awareness of better and more efficient solutions by the end users of the information systems, cause pressure for the development process to get the changes done more and more quickly.

Luckily, in the case organization, the willingness to improve the current ICT development process did not stop after Scrum was introduced. The organization is still trying to improve the processes and to find ways to gain flow efficiency with high quality results in the ICT development process. The CIO of the case organization listed the following goals for future ICT development. These goals aim to improve the flow efficiency of the ICT development process.

1. Faster development time for new individual services

2. Better transparency over ICT development projects
3. Improving the productivity of development work
4. Eliminating waste from the development work
5. Improving quality and performance in ICT solutions
6. Improved collaboration between ICT services and business units
7. Improved collaboration between owners of an information system, ICT services unit and supplier of the data systems.

Kanban method was introduced in the organization in one development project in which the development team wanted to use the principles of the Kanban method to control their development flow. The project was considered as successful since the requirements were delivered in time and the development flow was transparent for all the stakeholders. The potential of Kanban was recognized inside the case organization. The purpose of this study is to identify the potential benefits and drawbacks of Kanban in the case organization. This study aims to offer further reasoning to accept the principles of Kanban in all ICT development projects in the case organization.

1.3 Scope of the study

In this study, exploring Little's Law, the law of bottlenecks, and the law of variability approaches process efficiency. This study is focused on such ICT development processes in which the result is somehow visible for the end-user of an information system. Such processes that concern procurement of equipment for example laptops, updating production or test servers, or other supporting actions in which the data administration unit of Trafi is involved are excluded from this study. The focus of the study is to examine how the Kanban method can address to the key laws that have an effect on the ICT development process in the case organization.

1.4 Structure of the study

The rest of the study is organized as follows. Section two describes the case background and overviews the current software development framework in the case organization. Section three introduces the findings from the literature review. Section four gives an overview of what was studied and how the study was conducted. In section five, the results of the study are presented and in section six the results are reflected against the findings from the literature

review. Then these are used to form the improvement suggestions. The final section concludes the results and discusses the limitations of the study, and suggests recommendations for future studies.

2 Case Background

2.1 Finnish Transport Safety Agency Trafi

This chapter briefly introduces the structure of the organization and the main responsibilities of the different fields of operation. The current organization structure was formed in 2013. The organization is divided into four main fields of operation. In addition, strategy, communication and administration works form their own functions. Rail Regulatory Body is also an individual function in the organization. Each mode of transportation (aviation, maritime, rail, and road) has their own director. The organization structure is illustrated in Figure 1 below.

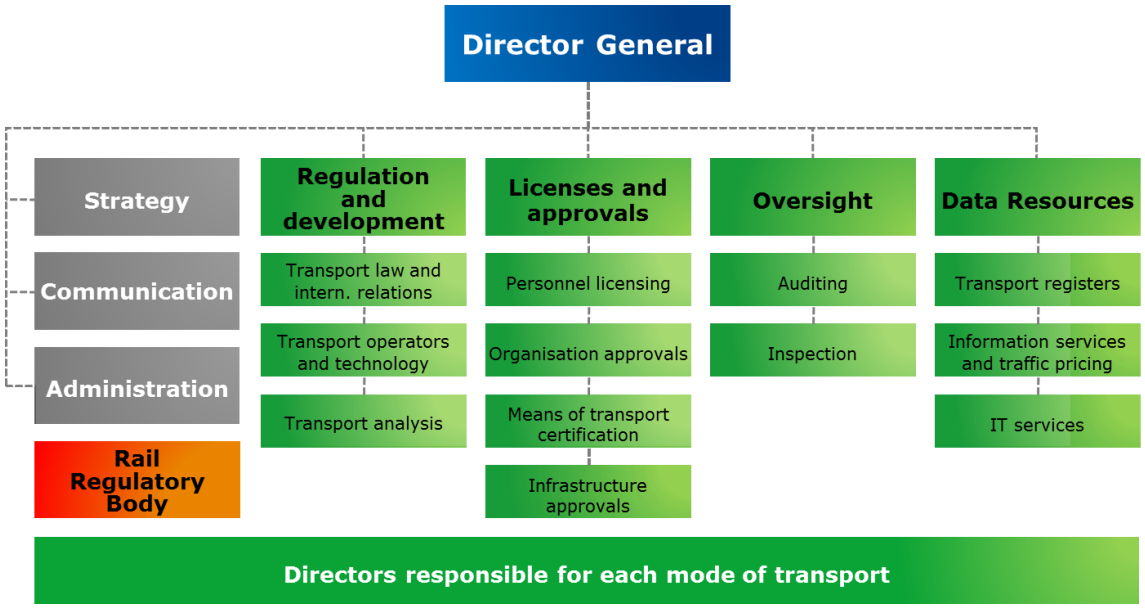

Figure 1: Trafi's organization (Trafi, 2014)

A field of operation is defined as a part of the organization that is responsible for a certain business function. There are four fields (divisions) of operation: Regulation and development, Licenses and approvals, Oversight, and Data Resources. Regulation and development division is responsible for transportation safety analysis, international relations and collaboration, and regulation actions of the agency. Furthermore, Regulation and development division coordinates the

development projects and project portfolio management in the case organization. Licenses and approvals division is responsible for producing license, approval, and qualification services. Oversight division carries out the oversight missions in each mode of transportation. Trafi oversees transportation safety and audits the parties that work in transportation business. In addition, Oversight division is responsible for developing the operation system for quality and environmental manners. Data Resources division is responsible for registration actions for each mode of transportation, consulting service providers, providing information and statistics, managing the agency's information and communication systems, other information technology and information security. Data administration department is part of the Data Resources division even though the department operates in every field of operation.

2.2 Trafi's ICT development

ICT development is a continuously going activity in the case organization. There are approximately ten software development teams either further developing existing information systems or building new information systems for in-house use or for cooperation partners, and even for Finnish citizens. Most of the development teams are located at Trafi's office in Helsinki, which supports the communication and cooperation between the development teams and business representatives. The sizes and lengths of the software development projects vary from couple month to several years depending on the requirement. Small projects include for example individual changes to existing information systems whereas large projects consists of broad changes to multiple information systems or even developing new information systems. Without exception, all the developers are subcontractors.

Since there are multiple definitions for an ICT development process, the term must be defined in the context of this study, and to clarify what the ICT development process means in Trafi's development environment. Figure 2 illustrates the examined ICT development process.

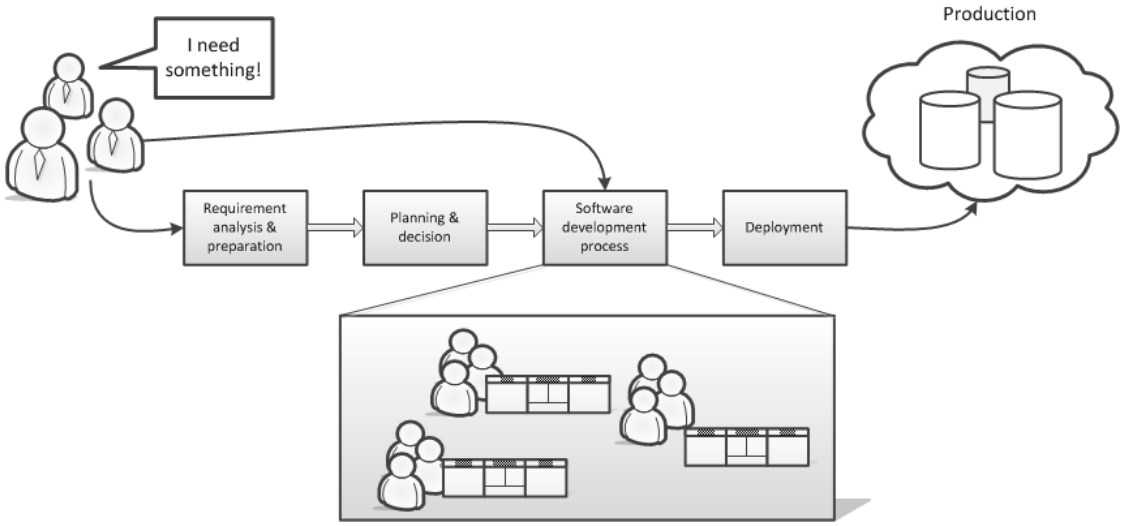

Figure 2: ICT development process

Simply, the study focuses on the above process which describes how an idea or a requirement is brought to a production environment. The whole process begins in the business units where the ideas and the needs arise. In a case where an idea or a need is related to an existing information system or the change does not have dependencies with other information systems, the requirement goes straight to the software development process. If the requirement is something that will most probably end-up as a new information system, or the requirement has multiple dependencies to other existing information systems, the requirement needs a more formal procedure. An analysis and preparation for the requirement, as well as planning and decision on whether the requirement will be implemented, is done before the idea or the need processes in the software development process. The whole formal procedure is presented in chapter next chapter.

Behind every software development process, there is a team which implements the business needs. Each software development team has their own development process, although the Scrum process is given as a guideline for the teams. The software development process consists of various steps such as designing architecture around the solution, implementing the requirement, and testing the implementation, including acceptance testing by the business unit representatives. Once the software development process is completed, the ready requirement is sent to the deployment process. The mission of the deployment is

to publish the results for the end-users, thus deploy the information system into the production environment.

2.3 Development Life Cycle Model

The Development Life Cycle Model (DLCM) describes the development work environment in the case organization. The goal of the model is to unify project procedure, how the projects are started and how they are managed from the beginning until the end. The model is divided into seven phases. At the end of each phase of the DLCM, a decision is made whether the development task is further developed or not. Thus, the model can be thought as a sort of a gate model. The model also offers a way to handle the development (or project) portfolio in the case organization. Even though the model describes the whole process from an idea to production, the main focus is on the first two phases where the development requirements are identified, analyzed and prepared. The DLCM process and its relation to The Adaptive Project Management Model (APMM) are illustrated in Figure 3. The APMM is further discussed in section 2.4.

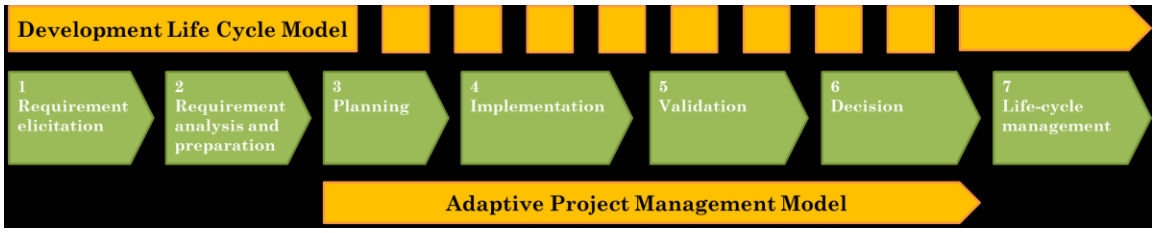

Figure 3: An overview of the Development Life Cycle Model

As illustrated in Figure 3, the DLCM is divided into seven phases. The final phase of the DLCM, life-cycle management, consists of multiple decision points where the life cycle of a completed system or service is validated.

2.3.1 Requirement elicitation

The goal of the first phase of the DLCM, requirement elicitation, is to identify and to collect possible development tasks. Typically the business units identify the development tasks. In this phase, development tasks are prepared and processed on an abstract level. All development tasks do not necessarily pass this

phase because prioritization is already done. Result of the requirement elicitation phase is a free form proposition of the development task to the operating committee. A development task is appointed to a person in charge for the next phases of the DLCM. The operating committee of the agency decides whether the development task will be further processed or not. If the development does not require action from other fields of operation, the operating committee of a certain field of operation can make the decision.

2.3.2 Requirement analysis and preparation

Requirement analysis and preparation phase aims to prepare the development task identified in the previous phase in a way that the operating committee of the agency or business unit can decide if the development task requires a project. The purpose of the requirement analysis and preparation is to study the dependencies to other ongoing projects and existing information systems. At the beginning of the phase, a person (usually a development director) responsible for preparing the development task is assigned. Also an owner for the development task is assigned from the business unit. Typically the owner of a certain system or a service is the director of the business unit. The owner is responsible for clarifying the needs of a development task. Each development task needs a project card, a schedule of work and other preparation material including dependency analysis, solution proposals, budget and resources. In addition, before the decision is made, an architecture card is also required. The architecture card is done in collaboration with the system architect. All these documents are produces in this phase.

2.3.3 Life-cycle management

The final phase of the DLCM is life-cycle management which begins when the actual development work of a project ends within the APMM and the information system or the service shifts to the maintenance and further development phase. The life-cycle management phase consists of controlling the service level, and the maintenance and further development of an information system or a service. Life-cycle management phase does not end until the information system or the service is shut down from the production.

2.3.4 KARKKI team

The KARKKI team is responsible that every new development project in the case organization is using and following the DLCM process model. The KARKKI team looks after and monitors that each development project follows the gates and decision points of the Development Life Cycle Model. Every project must follow the DLCM, and the KARKKI team is the decision maker at every gate of the process. In addition, KARKKI team assists the projects with the usage of the DLCM, in other words, consults and helps the projects to solve issues related to the model, when needed. KARKKI team is organized as a network. The network consists of advisers from different fields of operation. Furthermore, the KARKKI team is responsible for managing the development portfolio in the case organization.

2.4 Adaptive Project Management Model

Adaptive Project Management Model (APMM) is part of the DLCM, and it aims to standardize the existing project management in the case organization. APMM is designed as modifiable and lightweight as possible, so that every project started in the case organization could use the same project management framework. The model does not state how the project should be managed or how the project roles should be organized. However, the gates (decision points) are more restricted and they follow formal processing. At each gate the project requires approval from the KARKKI team and the project steering group in order to continue. APMM describes the implementation phase of a project. The model takes a stand on ICT development on a more detailed level than the DLCM. The software development methods and the roles are described in the model. Figure 4 below gives an overview of the APMM.

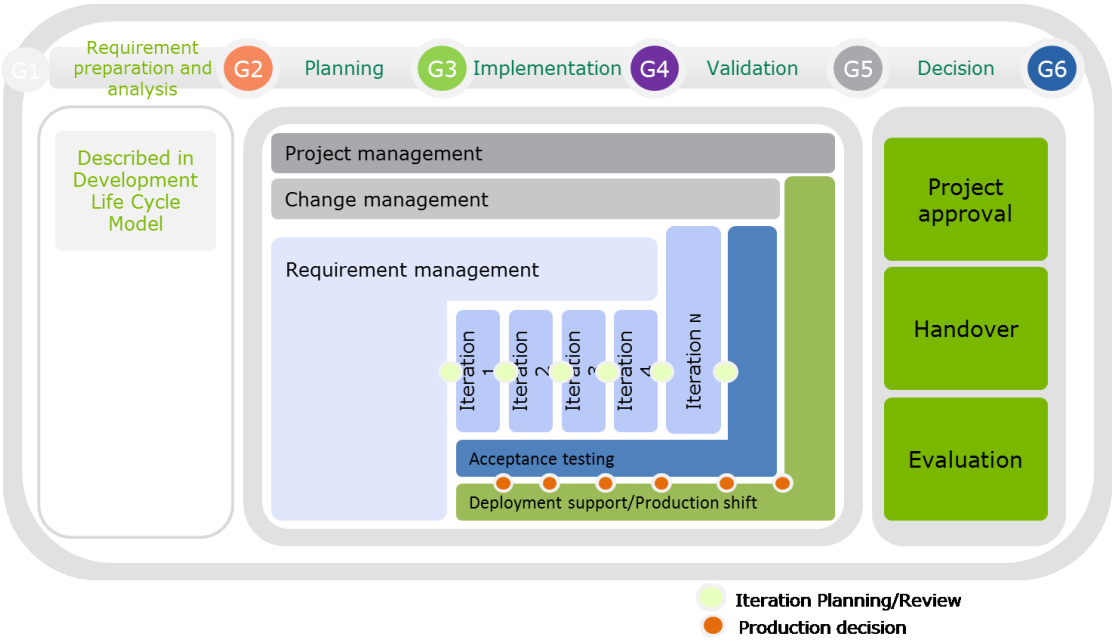

Figure 4: An overview of Adaptive Project Management Model

The Adaptive Project Management Model is divided into four phases: 1) Planning, 2) Implementation, 3) Validation, and 4) Decision. The phases resemble the traditional software development process phases. In the design phase, a team and a project plan are formed. Also an initial project backlog is created and prioritized. Tasks from the project backlog are processed in the Implementation phase. The Implementation phase lasts as long as the project backlog has tasks remaining, or when the schedule is completed. The goal of the Validation phase is to finalize the remaining development tasks and approve the end result of a project. The Decision phase aims to deploy the result of a project (e.g. information system) into production. Next, each phase of APMM is described in more detail.

2.4.1 Planning

In the Planning phase the development proposals receive a permission to be organized as a project. Trafi's board of directors or the board of field of operation owning the proposal makes the decision. Administrative planning and the content of a project are defined in the Planning phase. An initial time frame for the project is created. Also a release schedule is tentatively created. Content planning covers specifications for the information system or the service.

However, too detailed content specifications are not recommended, since the needs and requirements usually change during the project. Also the Product Owner is responsible for specifications during the project. The Planning phase results in a plan of the first implementation iteration with a detailed estimation of the amount of work.

2.4.2 Implementation

The second phase of the APMM contains the actual implementation work of an information system in an ICT development project. The implementation is carried out according to the release schedule. As the Figure 4 presents, the implementation phase is iterative, which means that the work is divided into small pieces and done in sequences. New features are released iteratively into the production environment. The Implementation phase continues until the business critical features of the information system or the service are implemented. Chapter 2.5 describes the work practices in more detail. The result of the implementation phase can be an information system, change to an existing information system or even a change to the existing work practices.

2.4.3 Validation

In the validation phase, the readiness of the information system or the service is validated. Traditional acceptance testing is organized in the validation phase. Even though the acceptance testing is typically conducted within each iteration in agile software development, the final acceptance testing is conducted by the business units for the whole system. The Release and final production deployment are done during the validation phase.

2.4.4 Decision

At the decision phase of APMM the project organization is taken down and the resources are discharged. Final changes and missing features are implemented into the system or service. Results of the phase are the final report of the project including lessons learned. Ending decision for a project is made by the project steering group based on the ending proposal made by the owner of the development task. After the decision phase, the project moves to back to the DLCM model and the life-cycle management phase begins.

2.5 Software development methods at Trafi

Previously the ICT development projects in the case organization have been following more or less the waterfall model (Royce, 1987). The transition from the traditional software development methods towards agile software development methods began during the fall 2012. Scrum was selected as the software development and management framework for the ICT development projects (Schwaber & Sutherland, 2013). The transition has been incremental, starting from transferring the existing maintenance work to an agile way of work. New ICT development projects have been started as Scrum projects since fall 2012.

Most of the roles in the ICT development projects are derived from Scrum. Roles derived straight from Scrum are Product Owner, Scrum Master, and Scrum team (Schwaber & Sutherland, 2013). The ICT Product Owner role was added beside the Product Owner role defined in Scrum. The ICT Product Owner role was taken into use to support the non-technical Product Owners. There are several other people involved in the development work among the Scrum team, for example system or application architects. Each project has their own steering group which consists of the Product Owner, the Project Manager, and business representatives such as head of a business unit. The Project Manager is typically responsible for one or more projects.

Table 1 below describes the roles and their responsibilities and tasks in ICT development projects in the case organization.

Role	**Responsibilities**
Project Manager	Project Manager is responsible for forming a project team and creating a project plan in the beginning of a project in collaboration with the Product Owner. Project Manager is in charge of the progress of a project (schedule, scope, and budget). Project Manager engages in iteration planning, content clarification and prioritization the project backlog with the Product Owner. In addition, Project Manager reports the status of the project to the steering group and the portfolio management group.
Product	Product Owner is responsible of collecting and sharing the business requirements to the Scrum team. Product Owner

Owner	also ensures that the business unit and Scrum team has common understanding of the business requirements. Product Owner is in charge of managing and prioritizing the project backlog in collaboration with the Project Manager. Product Owner accepts the output of iteration.
ICT Product Owner	ICT Product Owner acts as a technical adviser for the Product Owner. ICT Product Owner participates in project backlog management and prioritization. In addition, ICT Product Owner is the administrative manager for the Scrum team. ICT Product owner monitors the work of Scrum team. ICT Product Owner is responsible that the architecture follows the common architecture policy at Trafi.
Scrum Master	Scrum Master guides the work of a Scrum team. Scrum Master is responsible that the principles and practices of Scrum are followed. Scrum Master belongs to the project management group. Usually Scrum Master is a consultant.
Scrum team	Scrum team is the actual development team which is responsible for implementing the business requirements. Scrum team can consist of e.g. developers, designers, testers, and business advisers. Developers are without exception consultants. Scrum team is self-guided.

Table 1: Roles in ICT development

2.6 Deployment process

Deployment process refers to the actions needed in order to install a new version of an information system into a production or an acceptance testing environment. There are three parties involved in the deployment process: 1) Application supplier, 2) Production chief, and 3) Production service supplier. The deployment process itself is divided into three phases: 1) Software change, 2) Deployment commission, and 3) Deployment. Figure 5 overviews the deployment process in the case organization.

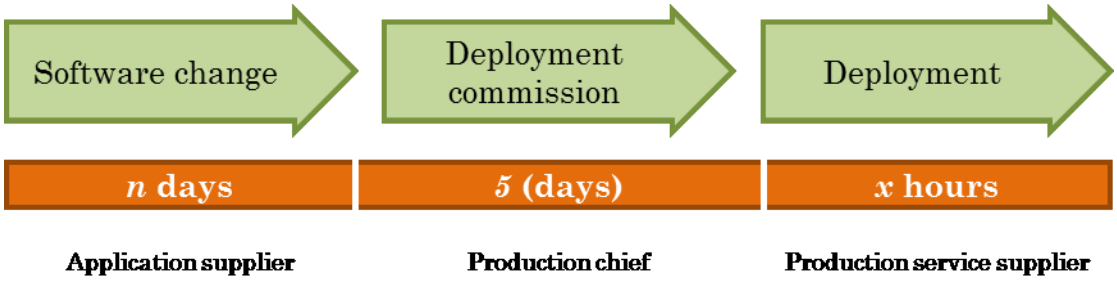

Figure 5: Deployment process

The application supplier is responsible for developing the requirements and implementing them into an information system, and the Production Service supplier is responsible for the deployment actions and maintaining the production environment. In practice, the Application supplier implements the software change in the first phase of the deployment process and delivers the new version of an information system to the Production Service supplier as a software package. In the case organization, for each information system, there is a person who is responsible for the production, called a production chief. In the second phase of the process, the production chief of a certain information system sends a commission to the Production Service supplier for each deployment of a new information system version. The deployment commission must be sent to the Production Service supplier five days in advance. In the provided final phase of the process, the Production Service supplier installs the new version of an information system into the production or the acceptance testing environment according to the instructions the Application supplier provides. Usually the deployment phase takes from ten minutes to a couple of hours depending on the software change.

2.7 Challenges in the case organization

Within the current ICT development framework in the case organization, the biggest challenges are long lead times and high dependency level. Even though the agile software development methods have been adopted, still the time spent to get the requirements from an idea level to the production is far too long, for various reasons. One of the major reasons for the long lead times is the fact that the internal resources do not have enough work allocation for the development work, as mentioned in the results of this thesis. Also the heavy development process model described earlier, as well as bureaucratic processes such as the

deployment process described in the previous section causes longer lead times for individual requirements. Simply because of the previously mentioned challenges, the development teams cannot publish new services and requirements in the production environment as often as it would be needed for fulfilling the business requirements, not to mention fixing possible bugs. Currently the lead time for an individual software change varies from four weeks to nine months.

Many information systems in the case organizations are somehow interdependent. Dependencies are problematic because a requirement in one information system causes changes in other information systems as well. Usually the implementation work is executed in separate development teams, and each of the development teams has their own separate goals (schedule, budget, scope). Thus, the development teams do no work in collaboration. Especially when there is lack of communication between the development teams, problems tend to occur.

3 Literature review

The goal of this thesis is to find techniques and solutions that help the software engineering organizations work in a flow efficient manner. Flow efficiency is one of the basic principles and well-covered topic in lean software engineering, especially in the Kanban method literature. The literature review is narrowed and focused on the Kanban method.

Basic books on the field of software engineering form the frame of the study. Modig & Åhlström (2012) covered the laws of process efficiency that are presented in this study (Chapter 3.1). Anderson (2010) on the other hand offered the principles and elementary knowledge of the Kanban method.

The main purpose of the literature review is to understand what the Kanban method is and how it could solve the flow efficiency issues in software development processes in general. Especially, recommendations and insights from case studies are interesting while studying the Kanban literature.

3.1 Process efficiency

In this chapter, the semantic differences and influences between resource and flow efficiency are discussed in order to understand what process efficiency means. Both resource and flow efficient approaches have certain benefits and challenges from the process perspective. In this chapter, the focus is not in a certain type of process, such as software development process, but the processes are examined on a general level. In many organizations, especially in the public sector, the work is often organized resource efficiently. A misleading thought is that an organization works efficiently when all its resources are working on full capacity (Modig & Åhlström, 2012). Flow efficiency, on the other hand, focuses on optimizing a value stream so that the flow unit, explained in chapter 3.1.1, receives value on an optimal manner. The three laws that each has a critical effect on process efficiency are: Little's law, the law of bottlenecks, and the law of the effect of variation on processes.

3.1.1 Flow unit

Flow efficiency is impossible to understand without introducing the key element in processes: a flow unit. Flow unit is defined as something that "flows" through a process; it can be material, information or people (Modig & Åhlström, 2012). For instance, in an automobile factory the flow unit is the automobile that is built in an assembly line. In other words, the flow unit is the element of a process that is getting the value out of the process from the end user's perspective (Modig & Åhlström, 2012).

In software engineering, at a high level, flow unit can be for example a user requirement, which flows through the development process and ends into an information system. The time and effort vary depending on the requirement. On one hand, a software feature may require little time and effort, for example in a case where the task only requires adding simple check on a user interface. On the other hand, the time and effort to implement the flow unit can be big for example, if the flow unit causes changes in multiple information systems or requires a whole new information system. Examining the flow units on a high level can be problematic because of the varying nature of the flow units.

If the flow units are examined on a more concrete level such as analyzed and well-defined development backlog items, the flow units are far easier to track and

measure. Each backlog item flows through a process, which is divided into multiple phases. Traditionally in software engineering these phases are: requirement analysis and preparation, design, implementation, and testing. However, the flow unit is not easy to define in the software engineering context. In this study, the flow unit is defined as a backlog item (or a feature) that flows through a development process.

3.1.2 Resource efficiency

Resource efficiency is the traditional form of efficiency in business environments. Resource efficiency is defined as an approach which focuses on utilizing the existing resources, such as people, machines, or information, as much as possible (Modig & Åhlström, 2012). Typically organizations strive to increase performance by keeping their personnel fully loaded. In practice this means that the organization ensures that its personnel have something to do all the time, even if the tasks do not benefit the value stream. A characteristic of resource efficiency is to divide development into small tasks that are processed by individuals or parts of an organization. Each individual or part of an organization is responsible for its own phase of a process (Modig & Åhlström, 2012). It is often well justified to use the resources as much as possible because otherwise the expensive investments would not make a profit and the money could have been used for something else.

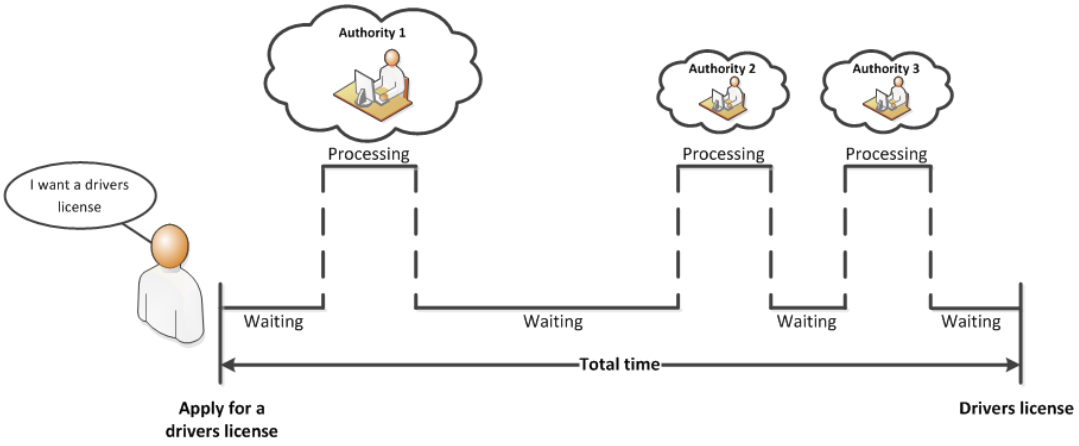

Figure 6: Example of a resource efficient process

An example of a resource efficient process is illustrated in Figure 6. In each step of the process, a different authority is responsible for processing the application,

thus the whole process is divided into small individual tasks. As there are most probably many people applying for a driver's license in a similar manner, it is likely that queues form in front of the authorities. In that case, the resources are well utilized and the process is efficient from the organization's point of view. However, from the applicant's perspective, the process is not efficient since there is much waiting time. More customer efficient process is illustrated in Figure 7 in the next chapter.

Often people perform multiple work tasks simultaneously even though it would be more beneficial for the system and the flow unit to finish a single task all at once than interrupt the task and starting a new one. Each interrupted task leads to a longer lead time, especially in process work. Delay affects the whole process (Modig & Åhlström, 2012). The next person in the process must wait for until the previous person finishes the task. Without tasks waiting in a queue, the next person(s) in the process would have nothing to do. This is the fundamental reason why organizations striving for resource efficiency prefer work tasks waiting in queues (Modig & Åhlström, 2012). From the viewpoint of the customer, waiting time does not add value.

According to Modig & Åhlström (2012), focusing too much on resource efficiency generates sources of inefficiencies. The sources of inefficient flow are: long throughput time, multiple flow units processed simultaneously, and need for restart per flow unit. Each source of inefficiency generates secondary needs which make the vicious circle even worse. In the driver's license example, the applicant might want to know at which point of the process the application is processed at the moment and then calls the first authority. The first authority must hire an assistant to answer the upcoming calls because the employees processing the applications do not have time for customer calls. A secondary need is generated. Once the amount of upcoming calls is increased too much, the assistant suggests that the company develops a web portal where the customers can see the status of the driver's license process. And another secondary need is generated. The applicant did not want any of this. All the applicant wanted was a driver's license.

3.1.3 Flow efficiency

Now that we have a basic understanding of flow units and resource efficiency, we can move onto discuss another form of efficiency, flow efficiency. Flow efficiency

is defined as follows: “Flow efficiency is the sum of value-adding activities in relation to the throughput time.” (Modig & Åhlström, 2012, p. 26). Every process has both value adding and non-value adding activities. Value-adding activities are those steps of a process in which the flow unit receives value. In other words, the flow unit is processed. Thus, something happens to the flow unit. Value is always defined from the customer’s perspective (Modig & Åhlström, 2012). In comparison, non-value adding activities are those in which the flow unit does not receive any value. Common non-value adding activities are queues in which the flow unit is waiting for. In software development, queues (referred as waste in lean software engineering) are typical — but often inevitable — non-value adding activities. A feature can wait for e.g. design, implementation or testing to begin.

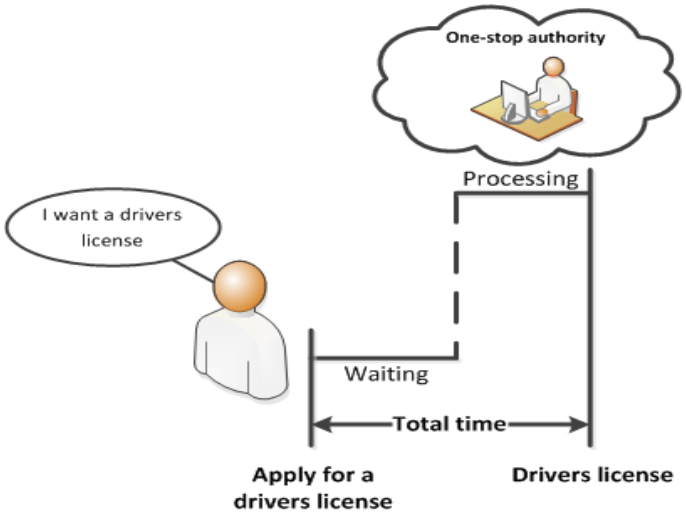

Figure 7: Example of a flow efficient process

Flow efficiency can be achieved by optimizing the value stream so that the barriers causing waiting or other delays for the flow are removed. In resource efficiency, the focus is on using the resources in a maximum utilization rate, even if the activity is not adding value. In flow efficiency, on the other hand, the resources are utilized when they are needed. In this case, the value-adding activity is performed at the right time and the next phase of a process can start the work without a delay. However, flow efficiency is not about increasing the speed of value-adding activities, but maximizing the density of the value transfer and eliminating non-value-adding activities (Modig & Åhlström, 2012).

An example flow efficient process is illustrated in Figure 7 above. In a flow efficient process, the authority that handles the applications concerns all the steps of the application process and does not send the customer to multiple locations where the next phase of a process is handled. There is possibly less waiting time for the customer since all the steps are handled in one place. There are no secondary needs that need to be fulfilled because of extra waiting. For example, there is no need for calling and asking about the status of the application since the applicant is aware of the process all the processing time. Modig & Åhlström (2012) uses our western health care system as an example to illustrate both resource and flow efficiencies.

The next sections introduce the laws that affect every process.

3.1.4 Little's Law

A law that has an effect in efficiency of any process is known as Little's Law. It is a mathematical model for calculating throughput time. Because of the simplicity of the law, it is widely used in manufacturing industry, for example to calculate throughput time of an assembly line (Modig & Åhlström, 2012).

In general, throughput time is the time that a flow unit spends in a process from a start-point to an end-point. Little's law defines that throughput time can be calculated by multiplying the amount of unfinished flow units in a process with the cycle time of a single flow unit (Modig & Åhlström, 2012). Cycle time is the time between when the work for a single work item starts and when the item is ready for delivery. Little's law is defined with the following formula:

$$throughput\ time = flow\ units\ in\ process * cycle\ time$$

According to the formula presented above, throughput time increases as the cycle time gets longer. Cycle time depends on different factors of a process, such as the length of a process or the available capacity (Modig & Åhlström, 2012). Both of these example factors can result in either a positive or a negative impact on the process. In some cases the process is naturally long, and it cannot be any shorter. When optimizing a process, after some point, the process cannot be further optimized. For a defined process, there is always an optimal throughput time. Human is one example of a limiting factor in a process, thus a person cannot work with unlimited speed for example. On the other hand, capacity affects

processes. If the capacity exceeds, cycle time increases, following with increased throughput time as the Little's Law shows. However, in some circumstances, especially when flow efficiency is desirable, it is acceptable that the capacity is higher than the demand since there is always variability in the process. Variability causes more demand for a capacity. Hence, flow efficiency sometimes requires extra capacity, which is overlooked in the resource efficient approach (Modig & Åhlström, 2012). The effects of variability in the processes are further discussed in chapter 3.1.6.

In addition to the effects of cycle time, unfinished flow units in a process have an influence on throughput time (Modig & Åhlström, 2012). Little's Law shows that the less there are unfinished flow units in a process the shorter the throughput time is (if the cycle time remains constant). Thus, unfinished flow units in a process cause longer throughput time for the whole process. In resource efficient organizations there is always a buffer in front of the resources. A buffer contains unfinished flow units. Buffers lead to increased throughput time, as Little's Law shows. Simply, the throughput time is shorter in a grocery shop where the cashier registers the goods from one customer than in a grocery shop where the cashier registers good simultaneously from multiple customers.

3.1.5 The law of bottlenecks

The law of bottlenecks, which has an effect on all processes, is a pretty obvious and familiar phenomenon. Bottlenecks are present in every process, otherwise the throughput of a process would be unlimited, and of course, in practice unlimited processes could not exist. It is recommended to eliminate bottlenecks in order to have better throughput of a process (Modig & Åhlström, 2012). However it should be noted, that once a bottleneck is eliminated from one place, it will appear somewhere else. Thus, eliminating bottlenecks should be an iterative and constant process in organizations (Anderson, 2010).

A bottleneck is a point in a process at which a queue forms (Modig & Åhlström, 2012). Thus, it is the phase in a process that has the longest cycle time, or in another words, the phase in a process that has the slowest flow. Accordingly, bottlenecks limit the flow in the entire process. According to Modig & Åhlström (2012), processes with bottlenecks have two key characteristics: 1) there is always a queue prior to a bottleneck, and 2) the phases after the bottleneck must wait for to be activated. The consequence of the second characteristic is that the

phases after the bottlenecks will work at a slower pace than they could. Thus, bottlenecks have a remarkable effect on throughput time in processes. Methods for identifying and eliminating bottlenecks in the software development processes are discussed in chapter 3.4.

Modig & Åhlström (2012) claim that there are two main reasons why bottlenecks appear in processes. First, the phases in a process must be performed in a certain order. To illustrate this problem, one can consider a cash counter in a grocery shop. A customer must place the goods on a counter before the cashier can register the goods. The goods must be registered before the customer can perform the payment. At least in a typical grocery shops, these activities cannot be performed concurrently. Secondly, variation causes bottlenecks in processes. Variation is present everywhere, especially if the process has people working in it (Modig & Åhlström, 2012). In the previous cashier example, people have different amount of goods, the goods are not necessarily food products and they require different procedure, people might pay with cash or a credit card. All these are examples of variation in the process. The following chapter discusses the impacts of variation in processes.

3.1.6 The law of variation

The third law affecting processes that is introduces in this thesis is the law of variation. Variation is something that deviates from a normal phenomenon (Modig & Åhlström, 2012). Modig & Åhlström (2012) divides the sources of variation roughly into three different groups: 1) resources, 2) flow units, and 3) external factors. Anderson (2010), on the other hand, divides the sources of variation into only two groups: external and internal. However, in practice, resources and flow units in Modig & Åhlström (2012) model are the internal factors as Anderson (2010) describes.

Resources cause variation in many ways. Machines may break, personnel may work at a different pace depending on the day, and for different persons it takes different lengths of time to finish a task (Modig & Åhlström, 2012). There are countless reasons why resources cause variation. Flow units cause variation because they are not often unifiable. Customers might want cars in different colors, houses build from different materials, a quality problem may cause rework and that affects the flow. Again, there are countless reasons why flow units cause variation. External factors are those in which the organization has no

direct influence. Even if the flow in a lunch restaurant is well implemented, conference guests from a hotel next door may cause huge variation compared to normal lunchtime by all coming at the same time.

Kingman's Formula shows us how throughput time is dependent on utilization. In another words, it describes the relationship between variation, resource efficiency (utilization), and throughput time (Modig & Åhlström, 2012). Kingman's Formula is illustrated in Figure 8.

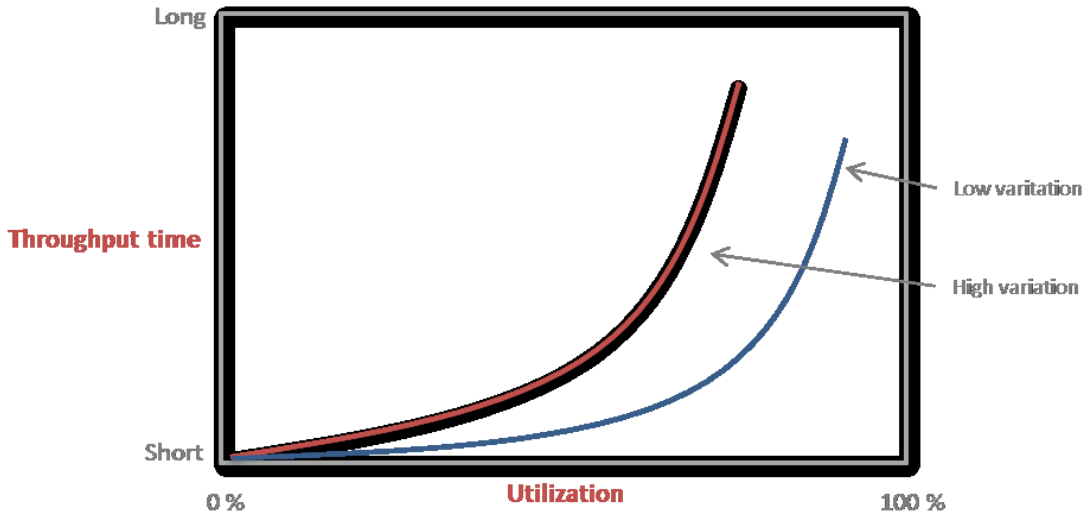

Figure 8: Kingman's Formula (Modig & Åhlström, 2012)

As illustrated in Figure 8, Kingman's formula shows that throughput time increases as utilization rate increases. Utilization rate is a measure of how efficiently the resources (people, machines, and so on) are utilized. The left curve in Figure 8 illustrates the relationship between throughput time and utilization rate on high variation. High variation leads to quicker increase of throughput time on lower utilization rate than in low variation. When utilization rate is constant, then the greater the variation in the process is the longer the throughput time (Modig & Åhlström, 2012). Furthermore because variation is included in every process, the focus should not be on maximizing the utilization rate, especially if an organization is focusing on reducing the throughput time. Resource efficient organizations think that their work and their processes are efficient but the efficiency is often only an illusion, as Kingman's formula shows.

As mentioned earlier, every process has inevitably variation in them, only the degrees of variation will differ (Modig & Åhlström, 2012). Variation results in a situation where the whole process does not work in an optimal manner. As Kingman's formula shows, variation leads to increased throughput time. Variation also has influence on the next phases of a process. If one phase is delayed because of high variation, the next phases will also be delayed. A delay is multiplied across the flow. Hence, organizations should focus on reducing variation in the processes in order to reach flow efficient process.

3.2 Kanban method

We have now been discussing about processes and examined the differences between resource and flow efficient processes. In this chapter, we move our focus from the process perspective to the second key area of this thesis, the Kanban method. At the same time, we come closer to software development processes.

The roots of Kanban are deep in Japanese thinking and Lean philosophy (Ohno, 1988). "Kanban" literally means a "signal card" (Japan Management Association, 1989). Originally these signal cards were used to indicate that more work can be pulled into the system. Today, in software development, kanban "cards" are used to represent work items or development tasks (Anderson, 2010). Cards are moving across a flow that is typically visualized on a whiteboard or such area. According to Anderson (2010), there are five properties that have been present in every successful Kanban implementation. The five basic properties that form the Kanban method are the following:

1. Visualize Workflow
2. Limit Work-in-Progress
3. Measure and Manage Flow
4. Make Process Policies Explicit
5. Use Models to Recognize Improvement Opportunities (Anderson, 2010)

Visualize workflow means to make the process behind the work visible for all the stakeholders. In practice, one can visualize the work flow simply by drawing the steps on a wall or a whiteboard. Visualized workflow aims to form a common understanding of the whole process between the stakeholders. Visualizing the workflow help teams to make decisions, to be aware of what the others are doing, and to identify bottlenecks in the process (Anderson, 2010). Visual control is also

used in manufacturing for eliminating waste in the production process (Japan Management Association, 1989). Limit Work-in-Progress (WIP) is the second key property of the Kanban method. Limiting the WIP helps teams to focus on finishing the tasks, instead of starting new tasks without control. Work-in-progress limits also direct the team to collaborate in order to finish the tasks. At the same time, the responsibilities move from individuals to the whole team. Measure and Manage Flow property states that process should be continuously improved to achieve flow efficiency. Any change should be measured and managed to set the improvement process to the right track. Some changes might have a negative effect on the process, and these changes should be identified. Process policies should be explicit to all the stakeholders of a process, including all the upstream and downstream stakeholders (Anderson, 2010). The continuous improvement is one of the core ideas behind Lean philosophy and the same concerns also the Kanban method (Japan Management Association, 1989). An organization should continuously pursue for perfection by recognizing models for improvements. These models can be for example Theory of Constraints or Systems Thinking (Anderson, 2010). The organization culture should be shifted to a direction where the customer and improvements are encouraged.

The Kanban method does not require any specific starting point from the organization that decides to implement Kanban, but existing processes and practices are required (Anderson, 2010). Thus, in software development the basis process before implementing the Kanban method can be anything from a traditional waterfall method to more agile form of software development, such as Scrum. Scrumban is one example of an adaption of the Kanban method implemented into Scrum environment (Nikitina, et al., 2012). However, Kanban is not a software development lifecycle process or a project-management process but a change-management technique (Anderson, 2010).

The next chapter presents the benefits of the Kanban method.

3.2.1 Benefits of the Kanban method

Anderson (2010) claims that by limiting the work-in-progress the team finds the way to balance the demand on the team against the throughput of their delivered work. Visibility provides chance to identify bottlenecks and variability in the process. Improved flow shortens the lead times and improves predictability and

due-date performance. Kanban help to build trust with customer and other stakeholders of a process (Anderson, 2010).

To support the claims presented above, Ahmad, et al., (2013) conducted a systematic literature review on the usage of the Kanban method and reported 16 benefits in software development projects achieved by using the Kanban method. The benefits, identified in the systematic literature, are listed in Table 2 below. The rightmost column in Table 2 indicates in how many studies the benefit was identified.

Benefit of the Kanban method	**Count**
Better understanding of whole	7
Improved software quality	6
Improved meeting of customer needs and customer satisfaction	6
Increased motivation of engineers	6
Improved communication/coordination between stakeholders/in team	5
Bugs were fixed more quickly, WIP made it easier to handle blocking in work	5
Increased software productivity	4
Problem solving (easy detection and removal of bugs)	3
Reduced batch size	3
Decreased time to delivery	2
Increased release frequency	2
Efficiently controlled software projects	2

Changes to requirements made welcome	2
Early feedback on features, without delays	1
No massive documents (limited to customer request)	1
Task approval from management not needed - approval gotten from customer in demos	1

Table 2: Benefits of the Kanban method (Ahmad, et al., 2013)

The main benefits of the Kanban method cover understanding the whole, improved quality, communication, collaboration, increased productivity and motivation, as they were seen in most of the studies (Ahmad, et al., 2013). As an example, Greaves (2011) report that the greatest benefit of Kanban is that it provides a mechanism to quantitatively measure the effects of any changes in the process. Measuring both demand and throughput of a customer support queue helped the organization to stay in balance by changing explicit policies (Greaves, 2011). The impact of Kanban on communication and on collaboration was studied in software engineering teams (Oza, et al., 2013). The study indicates that Kanban board is a good communication tool for development teams. However, the importance of a Kanban board in communication may become less important once the team gets to know each other and the communication is more direct and verbal. Kanban board may also help the team in collaboration (Oza, et al., 2013).

Benefits of the Kanban method can be seen in Anderson, et al. (2011) study. Anderson, et al. (2011) conducted software process simulations using WIP limits. Simulation was performed with 100 backlog features and 40 additional features added during the development. The report shows that when a WIP limit was defined, the whole development process became faster compared to an unlimited situation. When the WIP is limited, the workflow tends to be more constant than in the unlimited case. The report also claims that when developers work on every activity, not just for example on programming or testing, the length of the project tends to be shorter (Anderson, et al., 2011). Increased teamwork was also seen as a positive effect of Kanban implementation in the study (Seikola, et al., 2011). The study also shows that limiting the WIP supports teamwork.

3.2.2 Challenges of the Kanban method

We have now discussed the benefits of the Kanban method. However, as with any other method or model transition, implementing the Kanban method into an organization also has challenges. The Kanban method does not fit into every organization without doing iterative improvement actions. Ahmad, et al. (2013) identified challenges in using the Kanban method in software development. The challenges are listed in Table 3 below. The rightmost column in Table 3 indicates in how many studies the challenge was faced.

Challenge of the Kanban method	**Count**
Kanban cannot work alone but requires supporting practices	7
Hard to change organizational culture and philosophy	5
Lack of specialized skills and training	5
Collaboration and communication issues	3
Motivating staff to use new practices	3
Integration of Kanban projects with existing environments and processes	2
Hard to manage WIP	2
Hard to find a balance between the enforcing and caring roles	2
Kanban does not eliminate all waste	2
Hard to convince the top level management for the usage of new Kanban method	1

Hard to select tasks according to priority	1
Knowledge sharing	1
Misunderstanding of project goals	1
Need for guidelines for understand the process as a whole	1
Kanban may not fit well with existing corporate standards	1

Table 3: Challenges of the Kanban method (Ahmad, et al., 2013)

As noted in several studies, the Kanban method requires existing processes and practices in an organization (Ahmad, et al., 2013). However, the Kanban method does not state on top of which existing processes and practices the method should be implemented (Anderson, 2010). The Kanban method is designed to work along other processes and practices. Change resistance cause challenges in an organization when implementing a new continuous improvement culture with the Kanban method, even in top-level of an organization. People are used to work in a certain way and new way of work is even scary (Ahmad, et al., 2013). Convincing people to work differently was seen problematic. The situation has been noted in number of studies (Ahmad, et al., 2013). Specific skills and training was also noted as a challenge with Scrumban (Nikitina, et al., 2012). Managing WIP was seen problematic with lack of skillful personnel.

A flow-based Kanban process can sometimes be seen problematic and even shifting the process back to time-boxed iterations is possible, as Fernandes (2012) noticed. Developers suffered from a lack of goals when the iterations were removed, which also affected motivation. Fernandes (2012) report that the Product Owner could not participate in and be available for the developers as often as the flow-based development required. Time-boxed process gave the PO a chance to fit the planning meeting in his agenda (Fernandes, 2012).

Maassen & Sonnevelt (2010) faced several challenges in the Kanban method adoption. The challenges were noticed both in the team and the organizational level. Most of the challenges are in line with the challenges listed by Ahmad, et al. (2013). The challenges are listed in Table 4 below.

Team level challenges	Organizational challenges
Working within limits not needed: Personnel felt that they can do multiple tasks simultaneously without problems.	**Administrative processes:** Kanban was not used in all parts of the company which caused unnecessary and time consuming activities.
Limits reducing personal productivity: Personnel were used to keeping themselves as busy as possible while they were working and limiting work in progress disturbed that.	**Multiple locations:** Distributed work caused communication barriers within the organization.
Less than 80% allocation: Personnel that were allocated less than 80% for the team were seen more disturbing than helping.	**Separation in departments:** Defined responsibilities did not lead to team work. Personnel did not share the responsibilities but only ensured their own part of the work.
No input from sources: Personnel were unable to do the tasks if a key person was missing.	**Understaffed departments:** Certain departments were needed by all other department and they had more coming requests than they could handle. Thus, a certain departments were bottlenecks in the company.
Members approached directly: Personnel were working on multiple projects at the same time.	**Prioritization between multiple clients:** Multiple clients caused prioritization challenges, especially when there were multiple urgent rush jobs with the clients.
Tired of change: Personnel were tired of changing the	

approach how they do the work.	

Table 4: Challenges faced in an insurance company study (Maassen & Sonnevelt, 2010)

Lean production systems and lean software development talks about waste as something that interrupts the development flow (Japan Management Association, 1989). Waste, the non-value-adding activities, is definitely present in all software development processes. Ikonen, et al. (2010) identified waste in all the seven categories of waste in Kanban software development process: 1) Partially done work, 2) Extra processes, 3) Extra features, 4) Task switching, 5) Waiting, 6) Motion, and 7) Defects. However, the waste in the process did not significantly affect the success of the project (Ikonen, et al., 2010). Thus, zero waste is not a requirement for a successful Kanban project.

3.2.3 How to adopt Kanban

Greaves (2011) underlines the importance of self-organizing team in a Kanban process. However, right leadership is also required (Ikonen, 2010). A successful process requires constant interaction between stakeholders, communication, reprioritization, and following the common policies (Greaves, 2011). The author recommends a person outside the developers maintaining the relationship between the stakeholders and the development team. This enables consistent relationship and communication even if the line-up changes during the project. Greaves (2011) also claims that the team should not make accurate plans for the future since the plans will most probably change once the team learns more about the work. Teams should first start with a physical Kanban board since it is easier to adjust than an electronic board. According to Greaves (2011), the greatest benefit of Kanban is that it enables measuring the effects of any change in the process.

Maassen & Sonnevelt (2010) concludes their study and recommends that moving slowly with one team and sharing the results should start the Kanban adoption. The authors also underline that each team should create their own process and truly own the process. Furthermore, Maassen & Sonnevelt (2010), recommends that the teams should use their own kanban boards and update the status in a daily-basis. The board should be physical because it is easy to modify (Maassen &

Sonnevelt, 2010). Transparency and communication are factors that are seen critical in software engineering. Both Maassen & Sonnevelt (2010) and Neely & Stolt (2013) claim that the Kanban process should be transparent and clearly communicated between the stakeholders. Also, in order to succeed with Kanban process, the slowest parts should be automated, and the quality assurance should be fast and reliable (Neely & Stolt, 2013).

Kanban also requires training and education for all the stakeholders that are involved in the process (Nikitina, et al., 2012). Kanban process requires continuous improvement and high participation from the stakeholders such as managers, developers, and testers (Nikitina, et al., 2012). The author also recommends terminating Scrum iterations (Nikitina, et al., 2012). Responsibility and ownership should be kept in the teams (Seikola, et al., 2011). Seikola, et al. (2011) also highlights that active and skilled product owner help the Kanban process. Common open offices also have a positive impact on Kanban process (Seikola, et al., 2011).

3.3 Throughput time and Kanban

As presented in chapter 3.2, one of the five properties behind the Kanban method is limiting the work in progress (WIP) to achieve flow efficient process (Anderson, 2010). In software development with the Kanban method, limiting the work in progress is typically carried out in the development phases, such as design, implementation, testing and deployment, to indicate how many features can be in progress in a certain phase (Anderson, 2010). As presented in Figure 9, in the Kanban method limiting the work in progress is illustrated on the kanban board as a number on top of each development phase. A feature cannot move into the next phase of the process if there is no free slot for the task. Limiting the work in progress guides the development team towards collaborative work method and start finishing task instead of multitasking and starting but not finishing. For instance, if the tester is stuck with work, the developers can start helping the tester to unload the queue that is the testing phase (Anderson, 2010).

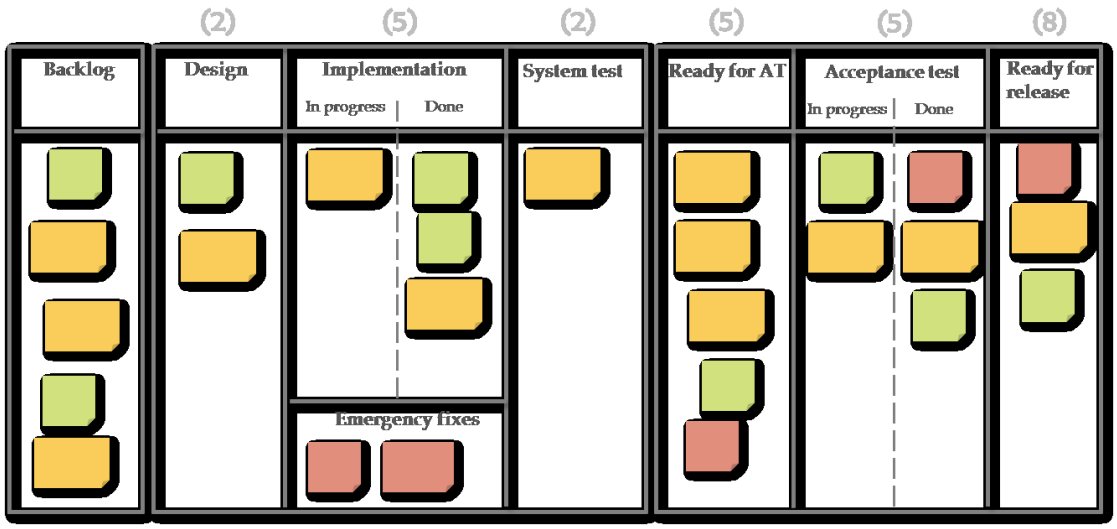

Figure 9: Kanban board representing WIP limits

Middleton & Joyce (2012) conducted a successful case study on the Kanban method at BBC Worldwide. They implemented a pull approach and limited the WIP in the development process using kanban boards and measured the effects over 12 months of operation. Middleton & Joyce (2012) report that after applying pull approach and limiting work in process, the variability of delivery of lead-time reduced by 47% from 70.7 to 37.3 working days ever the year. Time to deliver software features was also reduced by 37% from 22.8 to 14.4 working days. Middleton & Joyce (2012) also report that the development time was reduced even more than variability of delivery lead-time or software feature delivery time. The variance of development time was reduced by 78% from 30.5 days to 6.8 days. The development time fell 73% from 9.2 to 2.5 working days over the nine months (Middleton & Joyce, 2012).

Visualizing the workflow and measuring both lead times and cycle times is noted to be beneficial actions while continuously improving the software development process (Taipale, 2010). The report was from a small startup company in which the lead time from an idea to production is 8 days on average. Both visualizing the workflow and measuring the flow are principles of the Kanban method. Sjøberg, et al. (2012) showed that limiting WIP, implementing the pull approach, and abandoning cross-functional teams led to a situation where lead time for bug fixes declined from 12 days to five days, and for project backlog items from 14 days to seven days, after shifting from Scrum to Kanban (Sjøberg, et al., 2012).

Another study showed that limiting WIP and measuring lead times (mean task flight time) led to lead time improvement in which lead time dropped from 29 days to 10-12 days (Rutherford, et al., 2010). Limiting WIP also reduced lead time significantly in development at Software Engineering Professionals (Shinkle, 2009).

3.4 Bottlenecks and Kanban

As in any process, each software development process inevitably has restricting bottlenecks in the development flow. Typical examples of bottlenecks in software development are for instance a feature waiting for analyzing, an analyzed software feature waiting to be planned, the developing and testing the task itself or a developed and tested feature waiting for release (Anderson, 2010). Also a deployment or a production shift can be a bottleneck in software development process (Rutherford, et al., 2010). The point where the actual bottleneck is located depends on the team capacity, competence, development environment and many other factors (Anderson, 2010). The developers cannot implement features in an unlimited speed, as the testers necessarily cannot test new features at the same pace as the features are ready for testing. In a software development team where there are more resources for developing than there are for testing, the testing phase might become a bottleneck. This chapter discusses how the bottlenecks can be identified and eliminated from software development processes with the Kanban method.

3.4.1 Identifying bottlenecks

According to Anderson (2010), bottlenecks are possible to reveal simply by limiting the work-in-progress in the different phases of a process. When the WIP is limited, a bottleneck is identified in the phase that has too much load in it. Load can be seen as a queue in front of the process phase. On a kanban board, the cards stack in a bottleneck phase. Limiting WIP causes that a card does not move forward on a board. Personnel after the bottleneck phase cannot work in an optimal way since the tasks do not move to the next phase constantly, as Modig & Åhlström (2012) explained. For the whole system, the bottleneck causes even more trouble. Any extra time spent by a member of a development team is a sign of a bottleneck somewhere in the process (Anderson, 2010). For example, Rutherford, et al. (2010) noticed that task approval was a bottleneck when they added WIP limit to the approval phase of the development process.

Also, visible work flow help organizations to identify bottlenecks in the software development process. Kerzazi & Robillard (2013) discovered that visualizing the release engineering process and making it clear to all stakeholders made the bottlenecks visible and thus easy to identify. Visualization of the release engineering process was implemented on an electronic kanban board (Kerzazi & Robillard, 2013). Visibility made the bottlenecks apparent also in an insurance company study (Maassen & Sonnevelt, 2010).

Staron & Meding (2011) developed their own model for identifying bottlenecks in software development projects. According to Staron & Meding (2011), first the entire development flow should be clarified for the stakeholders, for example by using Value Stream Mapping (Staron & Meding, 2011). An overview of the flow helps the stakeholders to understand the main activities that are executed during the development. Secondly, an organization should focus on monitoring throughput time and queues in order to reveal bottlenecks in the development flow. Staron & Meding (2011) suggests that monitoring should be targeted in throughput instead of productivity. Monitoring productivity generates negative associations on people. Also, it is much harder to form a consensus on productivity among people. Throughput, on the other hand, is concrete and easily measurable. Organizations should also focus on monitoring capacity of a team or an organization. Managing capacity, either by increasing or reducing capacity, help revealing bottlenecks in a development flow. Staron & Meding (2011) also state that constraints that limit the flow are often seen as bottlenecks. However, a constraint, such as technology knowledge, is only limiting the capacity of an organization. The constraint should be removed from the flow. The final observation from Staron & Meding (2011) study refers to systems thinking. Even if the bottlenecks are identified and removed but the customer does not receive the product or service fast enough, the organization will fail. The flow should be considered as a whole, from an idea to a final product or service. Measuring and eliminating the bottlenecks should be performed across the flow.

3.4.2 Eliminating bottlenecks

Eliminating bottlenecks means removing barriers that are shrinking the flow in a process. In a simple example, if one removes the neck from a typical glass bottle, the water can flow out of the bottle much faster than it did before. However, this could lead to a situation where the flow of the water cannot be controlled. Eliminating a bottleneck from one point inevitably moves it to another

regardless of the process (Modig & Åhlström, 2012). Thus, to avoid sub-optimizations, the system should be considered as a whole, and the bottlenecks should be eliminated concerning the whole flow. Bottlenecks should be removed iteratively (Modig & Åhlström, 2012).

Anderson (2010) approaches bottleneck elimination through the Five Focusing Steps (FFS) framework. The framework is originally from Eli Goldratt's Theory of Constraints. Five Focusing Steps is the basis for continuous improvement in the Theory of Constraints. The Five Focusing steps are:

1. Identify the constraint.
2. Decide how to exploit the constraint.
3. Subordinate everything else in the system to the decision made in Step 2.
4. Elevate the constraint.
5. Avoid inertia; identify the next constraint and return to step 2 (Anderson, 2010).

The first step focuses on finding the bottleneck in the current value stream. However, FFS does not state how the bottlenecks should be identified. In the second step of the FFS framework, the potential throughput of the bottleneck identified in the first step is discussed and compared to the current throughput. The output of the second step is the decision how to approach the potential throughput. The third step encourages modifying the current value stream in a way that the potential throughput could be achieved. It is crucial for successful improvement action to think the value stream as a whole, not just a part of the system. In the fourth step the improvement is implemented in order to increase throughput and enhance capability. The key point of the FFS is in the final step since the continuous improvement is created when bottlenecks are always monitored and continuously eliminated from the value stream. As Anderson (2010) states, FFS is a tool for organization to create continuous improvement culture.

3.5 Variation and Kanban

Software development is knowledge work and thus variability is always present in the process (Anderson, 2010). The required features are not often homogeneous and cannot be put in a single form. Risk, effort, priority, and complexity are only few factors that might cause variance within the

development features. As software development is knowledge work, people complete the work, which is the ultimate cause for variability.

Anderson (2010) divides the reasons for variability in software development into two categories: internal and external factors. As described in chapter 3.1.6, variability results in more work-in-progress and affects lead times and predictability.

3.5.1 Internal sources of variability

Internal sources of variability are variations that are under the control of individuals, the team, or management (Anderson, 2010). Thus, the system can be directly affected by changes made by the people working in the team. Internal sources of variability can be managed by policies. In software development, internal factors causing variation are for example work item size, work item type, class of service, irregular flow, and rework (Anderson, 2010).

The first internal factor – work item size – that causes variance is easy to understand. Work item size varies depending on a feature. Anderson (2010) mentions that a user story is a one form to describe features in software development. The size of the user stories varies, as one can easily conclude. However, standardizing the acceptable size of a work item can reduce the variance. The team should focus on splitting the user stories as equal as possible to avoid variation. In software development, also the work item types vary. Work item types can be for example production bugs, refactoring, or upgrading. Work items can also be defined by sizes of work items, e.g. “epic”, “feature”, or “grain of sand” (Anderson, 2010). All of these are very different and they require different amount of work. However, identifying and diving the work items into different types, spread the variation and improve predictability in the system (Anderson, 2010).

Classes of service are assigned according to their business impact. Anderson (2010) claims that when an organization has a mix of classes of service, expedites requests randomize the work and cause increased lead-time mean and spread of variation. This reduces predictability of the system. Different kind of work items cause irregular flow which leads to unwanted variation. Each work item has different needs (size, complexity, risk, effort) unlike in most production line systems. Thus, the flow is irregular and has variation. External factors can also

create irregularity in the flow. Rework, known as defects or bug fixing, also causes variability (Anderson, 2010). Often, rework is unplanned and requires immediate action, e.g. when a bug is spotted in a production system. Organizations should pursue high quality in order to avoid rework, and thus variance.

3.5.2 External sources of variability

External sources of variability, on the other hand, are events that happen outside the control of the corresponding people. These events come from other parts of business or value stream, for example from suppliers or customers (Anderson, 2010). External sources of variability cannot be affected directly by internal policies. According to Anderson (2010), in software development at least ambiguous requests, expedite requests, environment availability, other market factors, and difficulty scheduling coordination activity are sources of external variability.

Ambiguously written requirements result in increased variability since often the team member is unable to complete the task without proper definition and the tasks will then be blocked until the requirement is comprehensible. Sometimes expedited requests, such as emergency fixes or important requirements, come up unexpectedly outside the internal process. Since these requests cannot be anticipated, predictability suffers, especially if the unexpected request has high priority and must be implemented. Environment availability problems are typical in software development (Anderson, 2010). For example, deployment into production environment is not necessarily under a control of the development team. The team must wait for until the other team has completed the deployment. The market may change unexpectedly, which causes re-prioritization of the features. And again, variability is increased, and thus the flow cannot work as planned.

3.5.3 Avoiding variability

Anderson (2010) claims that most of the variability in software development processes can be avoided by limiting work-in-progress and working in collaboration. WIP limits encourage teams to focus on lead time, throughput, and flow in collaboration. If there is idle time in the process, the team should resolve the issue together. For example, the development and deployment team should

work in collaboration in order to avoid idle time between development and deployment. WIP limit is a good control variable under high variability (Birkeland, 2010).

To avoid variability, a team can also limit WIP for each different work item type (Anderson, 2010). Limiting for instance the amount of simultaneous "epics" in progress, the team focuses on finishing the epic before starting a new one. Anderson (2010) claims that often it is much more time consuming to finalize an epic than a feature or a grain of sand. Anderson (2010) states that, e.g. the number of expedited requests in progress should be limited. Then the business unit is unable to rush features over the prioritization as they wish. Thus, only the most important features will be implemented and finished at the time all at once.

4 Research methodology

4.1 Research questions

The main reason for this study is to identify the challenges faced in the ICT development process in the case organization, as well as to find ways for the ICT development process to be more flow efficient. To achieve these goals, the current ICT development process and the challenges of it are critical to understand. This led to the first research question of this study. The second research question aims to gather scientific evidence of the Kanban method and flow efficiency theory. For the second research question, it was essential to define the process laws and their existence in software development processes. The third research question aims to identify ways to improve the current ICT development process with Kanban principles. Adopting the Kanban principles in one development project in the case organization and tracking the impacts approach the answer for the last research question.

The research questions are as follows:

RQ1. What is the current ICT development process in the case organization?
RQ2. How can the Kanban method help organizations to solve flow efficiency issues?
RQ3. How can the case organization achieve a more flow efficient ICT development process with the principles of the Kanban method?
RQ4. How has Kanban adoption helped the case organization with flow efficiency issues?

4.2 Literature review process

A literature review was conducted to answer the research questions, especially question two. In this section, first, the data sources and the search strategy are presented. Second, the selection process for the included papers is described. Finally, the inclusion and exclusion criteria for the papers are represented.

4.2.1 Data sources and search strategies

All the data searches were conducted in three electronic databases. The main electronic databases used in this study were:

- Scopus (63 hits)
- ACM Digital Library (32 hits)
- IEEE Xplore (3 hits)

The number inside the brackets indicate the amount of hits with the given search string. Relevant studies were searched using the following search strings:

1. Kanban
2. Scrumban OR Scrum-ban
3. Software
4. Development OR Engineering

These search strings were combined by using the Boolean "AND" operator. The searches were targeted into article title, abstract and keywords. Consequently, the paper had to contain all of the strings to be retrieved. The structure of the search queries was the following:

(1 OR 2) AND 3 AND 4

4.2.2 Selection process

The literature review started by searching the electronic databases with preliminary search strings, such as kanban. Searches were narrowed to find the most relevant papers. The main focus of search process was to find relevant papers discussing the usage, pros and cons of the Kanban method. Since the Kanban method was introduced to the public not until 2010, only papers published after year 2009 were included in the study. Papers written in English were included in the study, others were excluded. Also duplicate papers were removed.

Step 1	Search relevant papers from the electronic databases (with the given limitations)	n = 72

Step 2	Exclude papers on the basis of titles	n = 54
Step 3	Exclude papers on the basis of abstract	n = 32
Step 4	Identify relevant papers on the basis of introduction, results, discussion, and conclusion sections of the paper	n = 16

Table 5: Steps of the search process

As presented in Table 5 the first step resulted in 72 papers. The limitations explained above were used in this step. Search and selection process followed the formula presented below:

1. If title of the paper was relevant, the abstract of the paper was read (Step 2)
2. If abstract of the paper was relevant, introduction, results, discussion, and conclusion sessions were read through (Step 3)
3. The paper was included in the study, if the paper discussed practices, benefits, or challenges of the Kanban method in software engineering environment. Thus, the included papers were experience reports, lessons learned, case studies or simulation studies.

4.2.3 Inclusion and exclusion criteria

Papers were excluded from the study if the paper discussed kanban in other than software engineering environment, such as manufacturing industry.

The high-level inclusion criteria were as follows:

1. The paper is available
2. The paper is written in English
3. The paper has an author

In the second step the papers were included on the basis of titles. Paper was included in the study if the paper title indicated that the paper will discuss agile,

lean, kanban, scrumban, pull-system, or other process methods or philosophies. However, papers that were clearly out of scope were excluded. An example of an excluded paper based on the title was "Games for teaching software development". Also all the conference proceeding papers were excluded on the basis of title.

During the third step, papers were included in the study on a basis of abstract if the abstract indicated that the paper provides experience data on the Kanban method. Papers were excluded if the abstract indicated that the paper was not from the field of software engineering. Kanban is also used in manufacturing industry but these reports are out of the scope of this study.

In the final step of the selection process introduction, results, discussion, and conclusion section were read. A paper was included and used in the study if the paper discussed benefits or challenges of Kanban, or how Kanban affects throughput time, bottlenecks or variability. Also the papers discussing suggestions, lessons learned, recommendations, or insights were included in the study. If the paper was not an experience report, lessons learned, case study or simulation study, the paper was excluded.

4.3 Empirical research

Empirical research was approached with two different methods: interviews, and project management tool study. The goal of the interviews was to survey the bottlenecks and other challenges in the current ICT development process in the case organization. The interviewees were chosen from different areas of expertise in the case organization. Project management tool study was targeted to finding numerical data about the performance of each development team in the case organization.

First, this section introduces the structure of the interviews and how the interviews were conducted. Secondly the project management tool study is presented, explaining how the data was collected and which data was interesting in the context of this study. Finally, the four development teams that were researched in this study are briefly introduced.

4.3.1 Interviews

The goal of the interviews was to identify the challenges in the current ICT development process that is used in the development teams in the case organization. Also, the interviews offered valuable background information for this study, e.g. what is the current ICT development process. Interviews were semi-structured and consisted of specific questions from four categories. Although the interviews were more discussions-like than strict interviews following a pre-defined structure. Yin (2003) listed this kind of an approach as shorter case study interview. The interviews were not recorded and transcribed because all the interviewees did not represent the case organization, and the contracts did not allow recording. Also the recording was suggested and purposely left out because there were no specific plan for transcribing and listening the contents of the recordings (Yin, 2003).

Interview structure consisted of five themes which all were related to the development process: introduction, overview of the team, bottlenecks, variation, and concurrent work-in-progress (Appendix B). In the introduction, the interviewer presented the topic and context of the study. The objective of the introduction was to form a common understanding between the interviewer and the interviewee what is the interview about and how it is related to the study. Overview of the team covered questions about the team and responsibilities of interviewee in the team. Discussion about bottlenecks, variation and concurrent work-in-progress was the actual focus of the interviews. Each interview was planned to last for an hour but varied from 45 minutes to almost two hours.

A memo of an each interview was written down during the discussion. The memo was written on the interview structure document. The hand-written memos were validated at the end of each interview by going through the comments and the statements that were written down on the document in order to verify the validity of the discussion. Most of the interviews continued with further discussion after the validation since more ideas and thoughts were generated. After each interview the memo was written on a Word document and sent to the interviewee to double check the validity of the discussion. If there were unclear, additional or missing statements or comments in the document, the document was corrected based on the interviewee's feedback. This reduced the possibility for personal interpretations.

Interview	Role	Team	Date	Duration
[I1]	Head of KARKKI team	-	08.05.2014	60 minutes
[I2]	Head of PMO	-	13.05.2014	60 minutes
[I3]	Product Owner	Team 1	27.05.2014	90 minutes
[I4]	ICT Product Owner	Team 3	03.06.2014	60 minutes
[I5]	Scrum Master	Team 3	04.06.2014	60 minutes
[I6]	ICT Product Owner	Team 2	06.06.2014	45 minutes
[I7]	Scrum Master / Developer	Team 2	10.06.2014	60 minutes
[I8]	Scrum Master / Proxy-Product Owner	Team 4	24.06.2014	120 minutes

Table 6: Interviews

The roles and responsibilities of the interviewees varied inside the organization, as listed in Table 6. The interviewees were chosen from different organizational levels to fully understand the challenges in the ICT development work. First, the development directors who are responsible of the high level ICT development processes (DLCM and APMM) were interviewed. The reason for including them was to gain understanding of the whole development process and the challenges faced on an abstract level in the ICT development process. Second, the product owners, either the business product owner or the ICT product owner, of each development team were interviewed to fully understand how the work of each development team is managed, and what the key problems are in the implementation phase of the development process. Finally, the scrum master or a developer of each development team was interviewed. This offered tangible information about the problems that are faced in the actual implementation work during the development iterations. Apart from the two previous levels, these interviewees (scrum masters) were not representatives of the case organization.

4.3.2 Project management tool study

Apart from the interviews, quantitative data was gathered from the project management tools that are used in the development projects. The focus was on measuring the cycle times in the development process phases to illustrate possible bottlenecks. Atlassian Jira is used in most of the software development projects in the case organization. Other Atlassian products are used for example for documentation purposes. Development teams manage and prioritize their product backlogs in Jira. Additionally, some teams visualize the development process on an electronic Kanban board in Jira. The Product owners, ICT Product owners, project managers and other relevant business representatives participate in project management in Jira. All the data on the performance of the development teams was collected from Jira whereas an own calculation tool (Excel macro) was developed for collecting data on how many development projects are ongoing in the case organization, and in which DLCM phase each projects are in. The collected data and sources of the data are presented in Table 7.

Team	Source	Collected data	Time frame
Team 1	Atlassian Jira, control chart	Unresolved issues, average age of the issues, completed and not completed issues	08.03.2014 – 03.09.2014
Team 2	Atlassian Jira, control chart	Mean cycle time, median cycle time	01.05.2014 – 03.09.2014
Team 3	Atlassian Jira, control chart	Mean cycle time, median cycle time	01.05.2014 – 03.09.2014
Team 4	Atlassian Jira, control chart	Mean cycle time, median cycle time	01.05.2014 – 03.09.2014
Project portfolio	Project portfolio, Excel macro	Count of projects in the DLCM phases	09.05.2014 – 03.09.2014

Table 7: Collected project management data

Atlassian Jira has built-in reporting tools included in the project management setup. Built-in reporting tools cover sprint reports, epic reports, velocity reports, and different control charts, just to mention a few. In this study, the most used built-in report was control chart since it offered valuable data about cycle times for each development process phase. The control chart tool calculates the time each task spends in a certain process phase and draws a visual graph showing the progress. Measurements include both average (mean) and median cycle times, and in addition minimum and maximum cycle times for each development process phase. There is also a possibility to select the examination period. In this study, the calculations were from "all time".

The data that was collected from the project management tools was the average and the median cycle times of the development processes, divided into the process phases. In practice, the data represents the time (in working days) a software feature spends in the development process. Since the whole development process is typically divided into phases, the data also represents the time a software feature spends in a certain phase of the whole development process, for example how long a software feature waits for testing to begin and how long the testing takes. The data was exported into an Excel file in order to make the calculations easier. Also, once the data from the four teams were in the same Excel file, the comparison and unification of the data was easier to accomplish than in a case where the data would have been separated each in their own servers and project management tools. The data was a snapshot of the situation from the beginning of the projects until the point when the snapshot was taken.

In addition to existing project management tools, an Excel macro tool was developed for the purpose of this study. The Excel macro tool automatically goes through the project portfolio document that exists in the case organization and calculates how many development projects are in a certain phase of the whole DLCM process. The data about how many projects were in the different phases of the whole process is collected in order to understand if there are phases that are constantly overloaded and the projects get stuck in these phases. The data was collected over four-month period.

4.4 Teams

This chapter introduces the development teams that were studied in this thesis. Altogether, there were four teams that were examined. The teams were chosen

for the study for various reasons. First, Team 1 was known as a low-performing team, and thus the team and its challenges were an interesting subject for the study. Secondly, Team 2 was chosen because the team had similar premises with Team 1 but the team was performing more flow efficiently. Both Team 1 and Team 2 were maintenance teams for existing information systems. Third, Team 3, on the other hand, was developing a new service and had almost started from scratch. Thus, Team 3 had different premises than Team 1 and Team 2. Finally, Team 4 was known as a well-performing team and finding out the reasons for the performance deemed important and interesting for the study.

4.4.1 Team 1

Team 1 is responsible of maintaining and further developing several information systems that are used both internally in the case organization and by multiple cooperation partners. Co-op partners are for example car dealerships, car inspection stations and car importers. Information system development project started in 2002 and all the systems were in production in 2010. Most developers have worked together several years, apart from few person changes. Each developer in a team is familiar with one information system.

The development team consists of seven people: a Scrum Master, a designer, four developers and one test automation specialist. The Scrum Master looks after the development method and helps in requirement specification and analysis. The designer is responsible of requirement specification and analysis. In addition, the designer handles the manual testing in the team. Developers are responsible of implementing the backlog items, and testing the implementation. The test automation specialist takes care of the regression testing with automated user acceptance tests. Additionally, the test automation specialist helps the developers in developing automated tests.

4.4.2 Team 2

Team 2 is responsible for maintaining and further developing information systems that are used in individual's road licensing. There are seven information systems that are used in road licensing. The users of the information systems are from the case organization and from multiple cooperation partners such as driver's license examination receivers.

The development team consists of nine people: a Scrum Master (also working as a developer), seven developers, and one tester. The team has evolved among the years. The current team has worked together for six months. The development process is based on Scrum with slight modifications. For example, the team does not keep iteration planning sessions but plan the software features as needed. The work follows the principles of the Kanban method. It is notable that Team 2 has a very active Product Owner.

4.4.3 Team 3

Team 3 is developing a part of a new online service for Finnish citizens. The service is developed on top of an existing platform which was developed as a first part of the case organization's online service. The project started in spring 2014. The whole Scrum team is working together for the first time. However, couple of the team members has previously worked together.

The Scrum team consists of six people: a Scrum Master, a software architect, two developers, a tester, and a temporary user interface designer. As in Team 1, Scrum Master looks after the development method. The software architect is responsible for the application architecture, and works as a lead software developer in the team. Developers implement the backlog items, and participate in exploratory testing and test automation actions. The tester is responsible for all the testing activities and test automation, as well as organizing the user acceptance testing sessions.

The team is using the elements of the Kanban method on their daily working practices. An electronic kanban board is in active use. The team aims to limit the WIP in the future.

4.4.4 Team 4

Team 4 is developing a maritime register for boats and vessels. The information systems are used both internally in the case organization and by multiple cooperation partners. The development project started in the fall 2012 and the first services were in production in the spring 2014. The project is still in progress and new releases are published multiple times a year. The team members have changed during the course of the project except for the Scrum Master and a couple of developers.

Currently the team consists of eight people: a Scrum Master, six developers, and one user-experience specialist. The responsibility of each member of the team is similar with the other teams. However, the Scrum Master is helping the Product Owners in their responsibilities more than in other teams.

The work practices used in the team are mostly based on the Kanban method. The development team uses WIP limits and visualization of the development process. However, the working environment has forced the team to accept higher WIP limits than the ideal limits would be. Roles in the whole team are based on Scrum as in other development projects in the case organization.

4.5 Data analysis

The raw data gathered from the eight interviews (on the interview memos) was first gathered into one document. The data was divided into the two categories, bottlenecks and cause of variation that were the focus of the interviews. Each bottleneck and cause of variation was listed in separate tables and duplicates were merged. However, the source of each bottleneck and cause of variation is still available in the results. For each bottleneck and cause of variation, either an external or an internal source was dedicated, based on classification by Anderson (2010). The significance analysis of the bottlenecks was based on the frequency they were mentioned in the interviews. In other words, the more the bottleneck was mentioned in the interviews, the more significant it is. This analysis formed the interpretation of the interview data.

Project management tool study resulted in numerical data on the performance of the development teams. However, a big challenge when comparing the numerical results between the teams is that the development processes differ among the examined teams. Each team is using different development process phases. Team 1 has only three process phases whereas Team 4 has divided the process into eight phases. Also the working practices differ, for instance how the team uses the electronic project management tool. For example in one team the team members might update the status of a software feature as they finish the tasks or in other teams the updating is inconstantly performed. The latter case might end in zero minutes in the electronic system in one phase if the developer has completed both implementation and testing before she updates the status. All these reasons made the analysis of the data more complicated. It was not possible to make a conclusion of which team is performing the best.

5 Results

The purpose of the thesis is to examine the current ICT development process in the case organization and to identify the barriers that are preventing the process being flow efficient. This chapter presents the findings of the study. Results were gathered by interviews and by examining project management tools used by the teams developing information systems for the case organization. The results are presented in four categories. First, the bottlenecks that were found from the ICT development process are presented. Second, the causes for variation in the current process are explained. Third, the indication that there is concurrent work-in-progress is stated. Finally, the performance of each team is revealed from the data collected in the project management tools.

5.1 Bottlenecks

Bottlenecks in the current ICT development process were approached by conducting interviews. During the interviews, the most obvious bottlenecks were identified. The most distinct bottleneck in the current ICT development environment is the internal resource allocation, meaning the resourcing policy with the case organization's own employees. Support from the internal resources was seen as a critical factor in the development work by all the interviewees. However, the internal resources, either business or technical resources from the case organization, cannot support the development teams as much as the teams require. The same problem exists in both the high level process work (DLMC and APMM) and in the actual ICT development process, meaning the work in development teams.

5.1.1 DLCM level bottlenecks

As described earlier, the DLCM process is divided into seven phases. During the empirical research, it became clear that the second phase of the model, requirement analysis and preparation, is the biggest single bottleneck in the process. According to the interviews [I1] and [I2], most of the projects are stuck in the second phase of the DLCM process. The situation has been almost the same during the follow-up period which indicates that the there is a bottleneck in the phase two because the projects do not flow into the next phase of the process.

The challenge in the requirement analysis and preparation phase is the lack of resources to process the development requirements.

> "The second phase of the development life cycle management process is a bottleneck since most of the work is done in the second phase and it requires many resources which are not scalable in our environment, for example the system architect." [I1]

For instance, there is only one system architect in the case organization who can participate in filling the architecture card which is required before the development task can move on to the next phase of the model. Especially, when there are many development tasks concurrently in the requirement analysis and preparation phase, the work of the system architect is a bottleneck. However, before the model was in use, a project could start without any consideration and investigation about the dependencies. Compared to that situation, the requirement analysis and preparation phase sounds justifiable and reasonable.

Another formal step in the requirement analysis and preparation phase that is causing inefficiency for the process is filling a checklist which is needed before the development task passes the second phase.

> "We have noticed that the gates are possible bottlenecks in the process. For example the second gate requires filling a checklist which takes time to go through." [I2]

The checklist is a tool for KARKKI team during their decision process at gate 2. The checklist is filled by the person responsible of preparing the development task (assigned at the beginning of the requirement analysis and preparation phase). The checklist contains for instance information about the goals, stakeholders, dependencies, solution proposals, acquisition, schedule, resources and budged, benefits, risks and so on. At the time of writing this thesis, a single person handles the checklists, which inevitably is not enough work time allocation compared to the amount of work that is required. Especially, while the one person processing the checklist is not fully allocated to do the job but has other responsibilities.

5.1.2 Bottlenecks at the development team level

The bottlenecks identified during the interviews are listed in Table 8. Each bottleneck is listed as either an internal (I) or an external (E) bottleneck. Internal bottlenecks are those which are related to the internal actions and processes of the corresponding team. External bottlenecks, on the other hand, are those which the development team has no direct potential to act on, as Anderson (2010) described.

Bottleneck	Description	I/E	Interview
Resource allocation	Too low allocation for the responsible people leads to a situation where the software features wait in queues. Since the development process requires participation from the resources, lack of allocation causes bottlenecks in the whole development process. Examples of resources causing waiting time are: the system architect, the integration specialist, and business unit representatives. The time of the feedback loop increases as the resources has no time to take action.	I	[I1], [I2], [I4], [I5], [I6], [I8]
Acceptance testing	There is a communication gap between implementation and acceptance testing phases. Thus, the tasks wait for in queue between implementation and testing, until the communication from developers reaches the acceptance testing resources. Furthermore, deployment to the acceptance testing environment is performed only once a week (or less frequently) and thus the acceptance testing forms a bottleneck.	I	[I3], [I5], [I6], [I8]
Support functions	ICT-support is not working in close collaboration and at the pace with the development teams. For example, user accounts, access rights, and test servers come from the ICT-support and delivery times are usually slow. The development teams cannot configure the test or the	E	[I3], [I5], [I6], [I7]

	production environments when needed.		
Process knowledge	Case organization's internal resources do not have a common way to process the development tasks (or software feature) so that they are ready for implementation. This means that the development tasks are in some cases weakly defined and the developers cannot start implementing them. There is also lack of proactivity which leads to bottlenecks when tasks appear in the process at the same time. Also the internal resources do not understand the importance of their collaboration in the team work. Furthermore, the vision and the opinions about the requirement priorities differ in business units which leads to longer processing time. Product Owners may not know all the information and need support from the rest of the organization which leads to long processing times.	I	[I3], [I4], [I6], [I8]
Deployment	Production or test environment deployment process is outsourced to a third-party supplier. The deployment process is not automated and it is seen as a time and resource consuming activity. The production deployment is also an irregular activity, and requires random waiting from the third-party supplier. Furthermore, the deployment to the acceptance testing environments might be performed only once a week. Thus, the changes must wait for a week until they are available for end-users or acceptance testing.	E	[I6], [I7], [I8]
Other projects	A team or a project must wait for the output of other project(s). Information systems and the development work have multiple dependencies with other existing information systems or projects. A team or a project may need for example an interface or an integration description before the team can implement its own part of the	E	[I4], [I5], [I7]

	system.		
Part-time job	Development team members might work only part-time and the assigned tasks wait for until a team member finishes the task(s). Also the tasks might require peer-review by other developers and in some cases the developers have differing working schedules and the tasks wait for in a peer-review phase of the development process.	I	[I5], [I8]
System knowledge	Developers are specialized in a certain ICT system or a skill area. If a certain developer is missing, the tasks related to the skillset (ICT system or technology) of his or her are delayed. Furthermore, developers might need support from other developers because they do not have the knowledge on all technology used in a project.	I	[I3], [I8]

Table 8: Identified bottlenecks

In addition to resource allocation, there are multiple sources for bottlenecks in the ICT development work, as Table 8 presents. Acceptance testing was noticed to be a bottleneck in the development process. One reason causing the bottleneck is described earlier in this chapter: the person who is responsible of acceptance testing does not have enough allocation to perform acceptance testing when needed.

> "Since the customer is unable to test the new features in right environment when it would be possible, there is a bottleneck. We cannot install a new version of the software into the acceptance testing environment." [I5]

There is also a communication gap between the developers and the acceptance tester. Especially business people are not fully involved and aware of the tasks they need to do in order to promote the development process. This is a wide challenge in the development environment in the case organization as it was noted in multiple interviews. Another cause for the bottleneck is that new versions of software are installed into the acceptance testing environment only

once a week. Also the development teams cannot perform the deployment by themselves but a Production service supplier who is responsible for the production servers, as explained in chapter 2.7 performs the deployment. At the worst scenario, the new functionalities or the bug fixes wait for a week before they can be tested in an acceptance testing environment, and yet another day to be deployed in the production environment. Of course, this increases the lead time of a single development item.

> "All the supporting activities are outsourced. The test environments, as well as user accounts, roles and configurations are not in our own control which often is seen as delay. We need to wait other people to finish their work, and it seems that they do not understand the cause to our work if they do not finish their tasks right away." [I3]

Another wide cause that leads to a bottleneck in the case organization is the work in functions which support the development teams. According to the interviews, the reason for support functions being a bottleneck is the lack of visualization of the whole process and that they are not aware of the importance of their part of the whole development process. All the supporting activities such as creating user accounts and user rights are handled in one place which is clearly overloaded when there are multiple teams needing support. Other external variables causing bottlenecks are deployment policy and other projects. As discussed earlier, the development team cannot install new versions of software into both the acceptance testing and production servers by themselves. This was noted as one bottleneck since a deployment commission must be filled and delivered to the installing party five working days in advance, meaning that a single change, either a bug fix or new functionality, waits for five days before the change can be tested in acceptance testing environment, not to mention another five days that the change wait for before it is installed into the production environment. Because of the high dependency level in the information systems, almost every development team is dependent on one or several other projects. Delay in one project affects the others, often with harmful consequences. Lead time increases as the team must wait for an output of another project. As well as the deployment policy and support functions, other projects are also external sources causing bottlenecks.

Knowledge and system knowledge were divided as separate reasons for bottlenecks. Knowledge bottleneck relates to the internal work performed in the

case organization whereas System knowledge bottleneck refers to the skills of the developers. The development process begins by identifying the requirements. This part is often quite straightforward but the challenges occur in the requirement analysis and preparation. Several people in the business units perform requirement analysis and preparation and often even the product owner of a team is not aware of all the requirements that are in progress. When many people are working on the same requirement, the lead time increases and this part of the process has been noticed as a bottleneck in the development process. System knowledge, on the other hand, causes a bottleneck once the only developer knowing a technology or an information system is absent. Teams have not often shared the system knowledge among the team members but all members have their own skills. As one can imagine, once the required skill is missing, the tasks related to that skill are delayed. In most of the cases, missing system knowledge causes bottleneck in the development process and increases the lead time. Sometimes even the team work causes bottlenecks. For instance, if there is a peer-review for each task and the peer-reviewee has no time to perform the peer-review at the time needed.

In summary, there are various reasons for bottlenecks in the development process. However, one of the biggest reasons for bottlenecks is the resourcing policy. The development teams need support from business people but they are not allowed or simply cannot find the time to work with the team enough because they have other responsibilities in the organization. Also, old contracts with the suppliers cause challenges in agile work practices. As the development teams are not able to configure and deploy the new versions of the software into the acceptance testing and the production environments, much time is wasted filling out the official forms and waiting other parties to complete their part of the process.

5.2 Variation in development

Another theme in the interviews and in this thesis was to identify sources of variation that are present in the current ICT development processes. Most evident sources of variation noticed during the interviews were quality problems and work item sizes.

There are several factors that cause variation in the current ICT development processes. The different factors causing variation are listed in Table 9 below. The

sources of variation are divided into two categories: internal (I) and external (E). Each source of variation is explained in Table 9 in the Description column.

Variation	Description	I/E	Interview
Work item size	Sizes of the development tasks or software features vary. The development tasks require different amount of effort to implement. The size can vary from a minor typo fix on the user interface to a whole new feature requiring changes in multiple information systems.	I	[I1], [I3], [I5], [I6], [I7], [I8]
Quality problems	Defects (or bugs) cause rework which affects the development process. Some defects might need instant attention, especially when they are detected in a production environment and the bug must be fixed instantly, thus the development process is interrupted. Furthermore, the quality problem might cause instant actions in a team during the development work e.g. if the software build is broken.	I	[I5], [I6], [I7], [I8]
Work item type	Work item types vary. Software features might be regular user stories, technical problems, bugs, and so on.	I	[I1], [I5], [I7], [I8]
Changing requirements or priorities	Requirements or priorities that change in the middle of development process cause variation. Requirements might changes because the corresponding law change or the needs of the business change.	E	[I4], [I7], [I8]
Resources	Resources produce different quality. Possibly, a developer leaves the team and a new resource must be hired to fill the missing resource. Different developers produce code differently and thus the quality varies.	E	[I3], [I4], [I7]

Technologies	Either existing technologies or new technologies vary, and require training. Existing, perhaps outdated or incompatible technologies that are used in other information systems and services cause variation to the software features. New technologies require closer attention and training to produce the same quality than with a well-known technology.	I&E	[I4], [I5]
Schedule or budget	Schedule or budget changes during the development when the existing requirements might need modifications. Some features might need to be left out from the original requirement.	E	[I4]
Third-parties	Different priorities and non-transparent processes in other organizations cause variation in the internal development process. Third-parties are the actors that work out of the power of a development team.	E	[I6]
Unclear requirements	The quality of requirement definition varies which causes rework especially if the team must return to a requirement and fix the errors.	I	[I6]
External demands	External demands such as information requests, integration guidance, and all the other demands which are not originally from the team backlog cause variation in the development process.	E	[I8]

Table 9: Identified causes of variation in development teams

5.3 Work-in-progress

One of the goals in this thesis was to explore the ICT development process in the case organization from the simultaneous work perspective. Interviews revealed that both on the high-level process and the project level and in the development

teams there is much concurrent work-in-progress. Figure 10 illustrates the count of development tasks (that are treated as projects) in each phase of the DLCM during the follow-up period. Different colors represent phases in the DLCM.

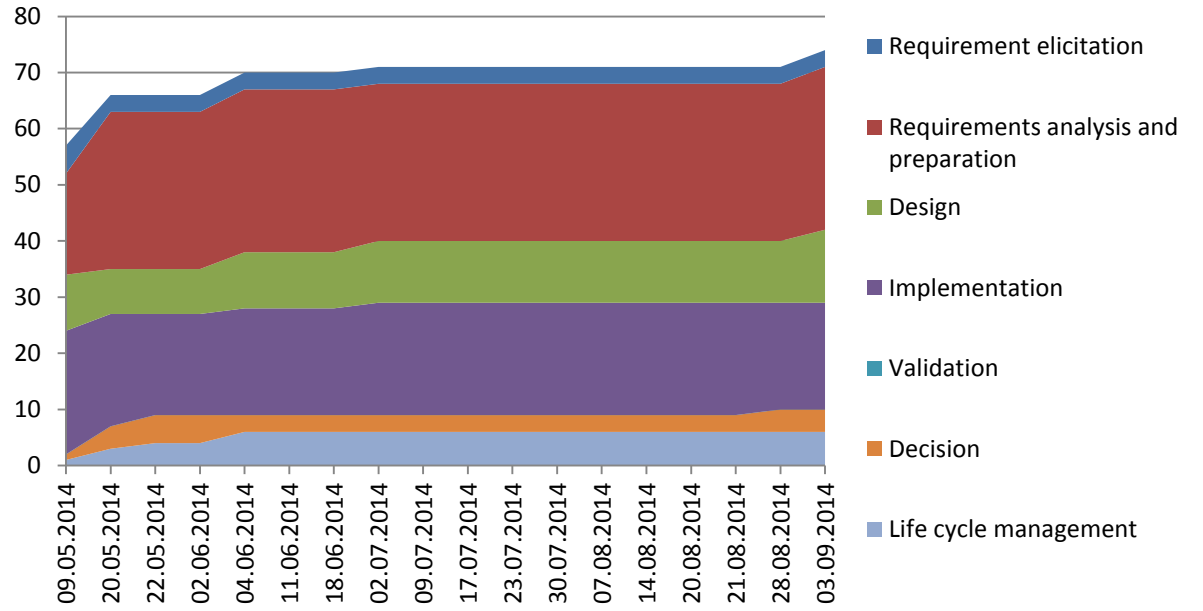

Figure 10: Cumulative flow diagram (CFD) of the DLCM phases over the follow-up period

We can see from Figure 10 that not much progress occurred in the process during the follow-up period since the amount of development tasks in the phases were more or less constant. If we consider the phases individually, we could understand Figure 10 either so that the amount of development tasks in a certain phase is almost constant because the circulation of development tasks is constant or so that the development tasks are not moving forward in the process. If we consider Figure 10 as a whole, we notice that the amount of development tasks in the last phase of the process is not increasing. This fact supports the claim that the organization is not finishing the development tasks but the flow is rather stuck and the development tasks are not moving in the process. On the other hand, in the development teams the individual tasks are moving from sprint backlog to ready for release, as seen in Figure 11. Ready for release stack is increasing over time. This indicates that the team is finalizing tasks. However, as the phase name indicates, in that phase the development tasks are only ready

for release, not ready in a way that they are available in production environment for end users.

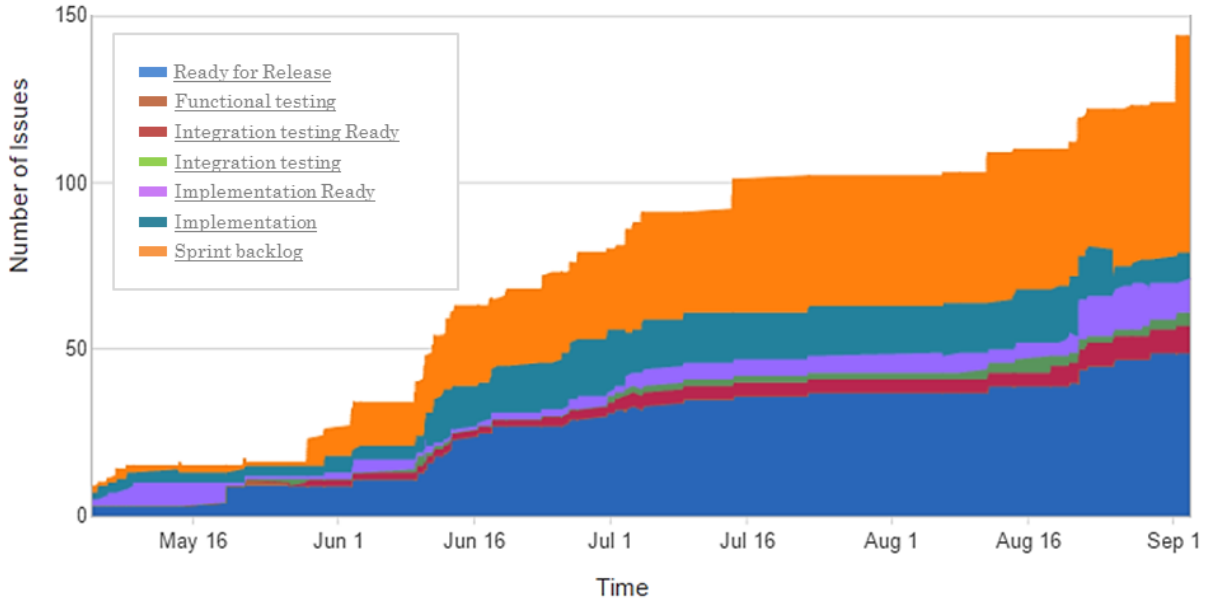

Figure 11: Team 3 CFD

At the moment, the amount of development tasks in progress is not limited. As illustrated in Figure 10, the Requirement analysis and preparation, and the Implementation phases include the most ongoing development tasks in the case organization. Respectively, the Validation, the Decision and the Requirement elicitation phases have the least simultaneous ongoing development tasks. Table 10 presents the situation at the end of the follow-up period.

Phase	**Count**
Requirement elicitation	3
Requirements analysis and preparation	29
Design	13
Implementation	19

Validation	0
Decision	4
Life-cycle management	6
Total	**74**

Table 10: Total count of projects ongoing 03.09.2014

There were in total 29 development tasks in the Requirements analysis and preparation phase at the end of the follow-up period, and the number has been nearly constant during the follow-up, as illustrated in Figure 10. The total count of ongoing development tasks was 74 at the time the follow-up period ended. At that moment, 39.1 percent of the ongoing development tasks were in the second phase of the DLCM. In the Implementation phase, the amount of development tasks varied between 18 and 22 during the follow-up period. The Design phase included 13 development tasks, and the count has been increasing during the follow-up period. As we can see in Figure 10, the organization was unable to finish the development tasks which were started. The last phase of the process, the Life-cycle management, did not significantly increase during the follow-up period.

> "In our team we process multiple tasks simultaneously and sometimes there are many unfinished tasks waiting for something. We must wait for output from other teams and we cannot do our part of the work before they are finished." [I4]

Also, the development teams implement multiple software features simultaneously, according to several interviews. The team members may start processing new tasks if for example the previous one gets interrupted. In some cases, the team is unable to finish the tasks because there is a dependency to other implementations, or the ongoing task needs better and further definitions by other parties. The team must wait for an answer from somewhere else, e.g. from a business representative before the team can implement the requirement. This situation also causes bottlenecks as discussed in the previous sub-chapter. Team 2 and Team 3 are striving for finishing the software features as fast as possible through the development flow. Team 2 has seen limiting the WIP

challenging because there is lack of competence in the team, and thus for example database migrations are processed outside the team. Hence, there are work tasks that are outside the control of the team. Team 3, on the other hand, have not started limiting WIP since the project just started and the team is focusing on finding the right working practices before limiting the WIP.

5.4 Performance

The project management tool study resulted in tangible numbers about the performance of each development team studied in this thesis. One possibility is to compare the average cycle times in the development teams. Table 11 below represents the mean cycle time of the whole process, and the mean cycle time for the implementation phase in the four teams examined in this thesis. Mean cycle time was calculated by counting the total time spent in the development process (containing all the development phases). Mean implementation time represent the time spent in "coding" phase of the process in which the requirement was implemented. We can notice that the average implementation time does not vary as much as the average cycle time of the whole process. This indicates that the processes differ from each other more than the actual coding of a single software feature.

Team	Mean cycle time	Mean implementation time	Process phases
Team 1	n/a	7,4 days	3
Team 2	57,3 days	4,5 days	7
Team 3	18,9 days	3,9 days	7
Team 4	62,8 days	5,0 days	8

Table 11: Overall performance of the teams

The following sections will introduce the data per each team that was studied in this thesis. For teams 2, 3 and 4, the performance data tells the cycle times (both mean and median) in days and how many percent of the time is spent in a certain process phase. Cycle time was collected in these three teams because at the

beginning of the study it seemed to be a comparable measure. However, as discussed in section 4.5, the comparison is not that simple. For team 1, these cycle time calculations could not be done since the process was only divided into three phases (Backlog, To Do, Done) and thus the numbers were not reliable. Also Team 1 has not been using the project management tool actively. The status of the development work was not up-to-date during the sprints.

5.4.1 Team 1 performance

Despite the challenges gathering data from Team 1, this sub-chapter aims to offer performance indicators of the team. For team 1, the project management tool study provided numerical data about the unresolved issues in the development process, and the average age of a software feature spent in the process. Figure 12 illustrate the situation over 180 days. As Figure 12 clearly shows, the team does not concentrate on finishing the task since the average of age of the issues is increasing.

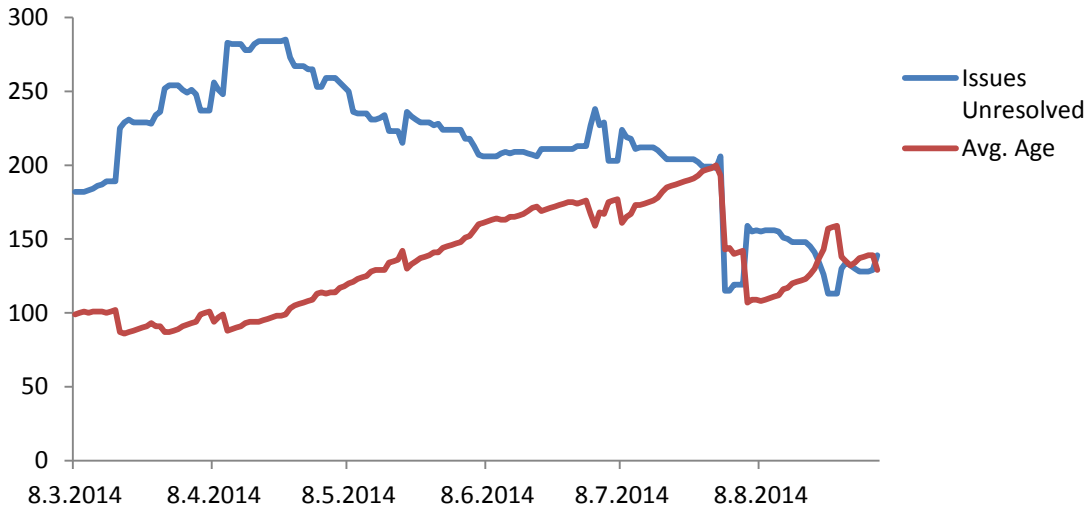

Figure 12: Team 1 unresolved issues and average age of the issues

The average age (the time a software feature has been registered in the project management tool) of a single software feature has been roughly between 100 and 200 days. At the same time, the amount of unresolved issues has declined from approximately 280 issues to 130 issues during the follow-up time.

Another interesting statistics from team 1 (see Figure 13) indicates that the team continuously takes software features into the sprints and never finishes all the features during the sprint. The amount of Not Completed tasks has even been increasing during the follow-up period, as illustrated in Figure 13.

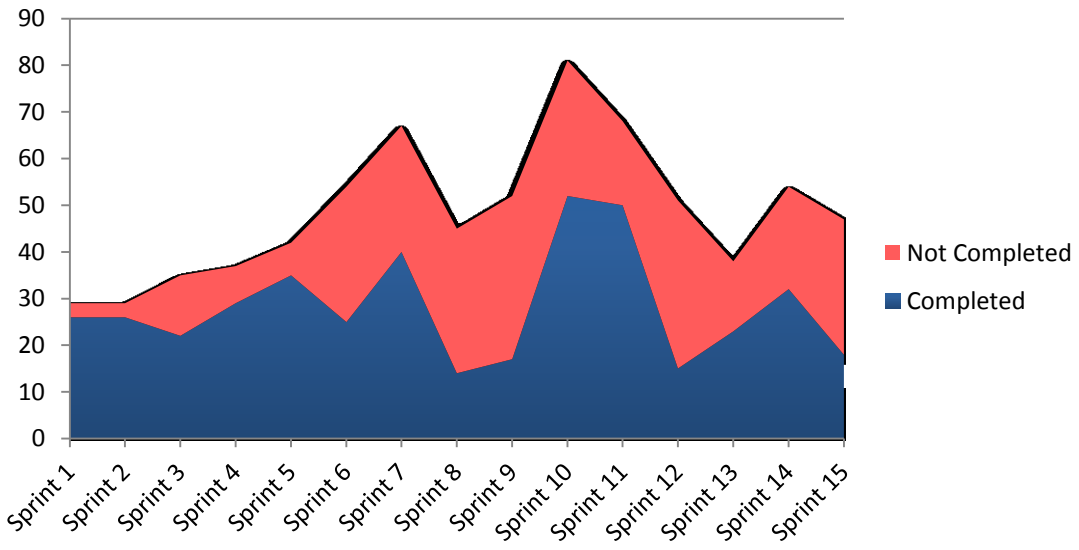

Figure 13: Cumulative flow diagram of completed and not completed issues

5.4.2 Team 2 performance

Measurements from Team 2 are presented in this sub-chapter. Figure 14 shows two measurements from the data: mean cycle time, and median cycle time. Mean cycle time is, as the term defines, the average time that a software feature is processes in a certain phase of the development process. Median cycle time, on the other hand, is the numerical value separating the higher half of the data from the lower half, thus it is the center point of the data. In some sense, median value gives more reliable results of the reality since in the mean value the highest or lowest values have more importance in the number.

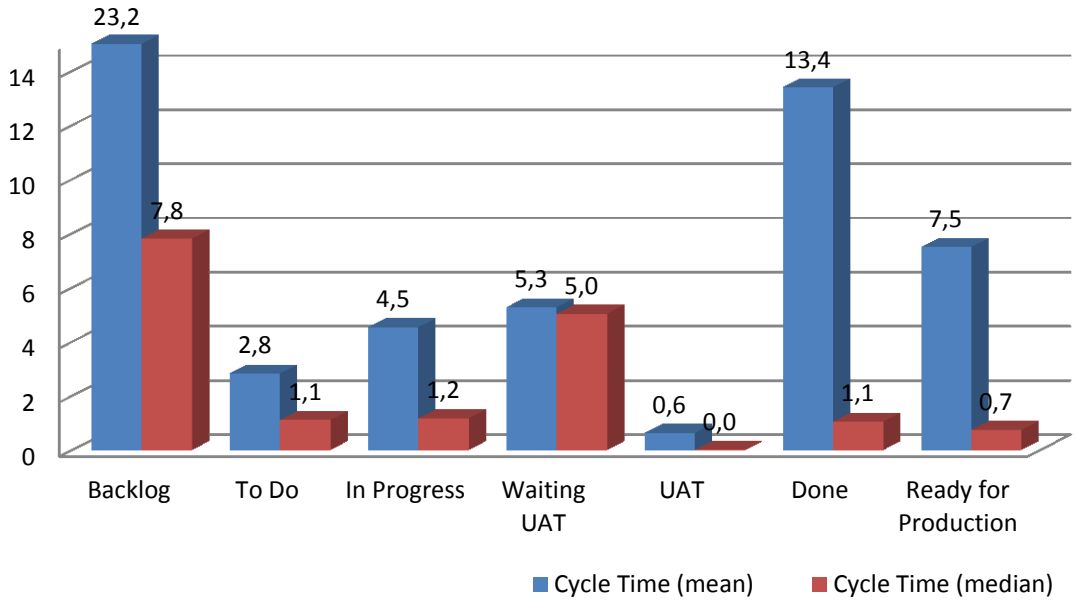

Figure 14: Team 2 mean cycle time in the development process in days

In the sixth interview, the User Acceptance Testing (UAT) phase was claimed to be a bottleneck in Team 2's development process. As illustrated in Figure 14, UAT consists of waiting the UAT to start the actual testing. The total time (average) for UAT is almost 6 working days (5.3 days + 0.6 days). UAT then takes approximately 33 percent more time than the actual implementation work (In Progress). Thus, UAT takes approximately 10 percent of the whole processing time. The statistics support the claim that UAT is a bottleneck in the process, although the biggest challenge is not in the UAT phase. Software features wait for in the Done phase approximately 13.4 days before they are accepted and move to the next phase. In average, it takes another 7.5 days before a software feature is deployed into the production environment.

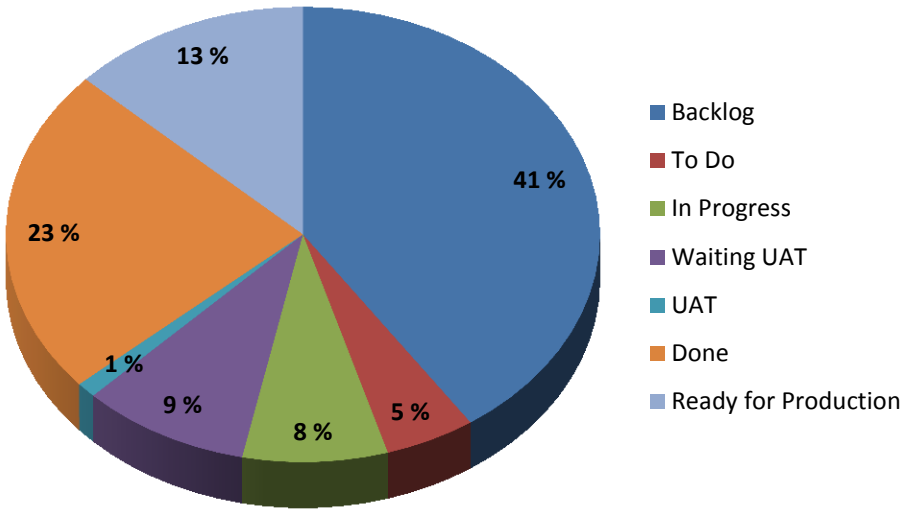

Figure 15: Team 2 mean cycle time in the development process in percent

Results show that features are waiting for to be released approximately 36 percent of the time (Done + Ready for Production phases), as Figure 15 shows. This is the result of the current release policy. The development team is not able to deploy the new version into the acceptance testing environment or the production environment, and the deployments are performed only once a week. As one can notice, lead-time of a single software feature increases because of the long Backlog queue (41 percent of the time). In practice, the software features can get through the Backlog faster than the statistics show if prioritization is done. The Backlog phase represents the whole product backlog, not a single sprint backlog as Team 3's process shows.

5.4.3 Team 3 performance

The data represents the same information as Team 2 measurements (described in the previous sub-chapter). The development process has seven phases as in Team 2 but the phases differ from one another. However, the output of the development process is the same; new features or bug fixes in the information system.

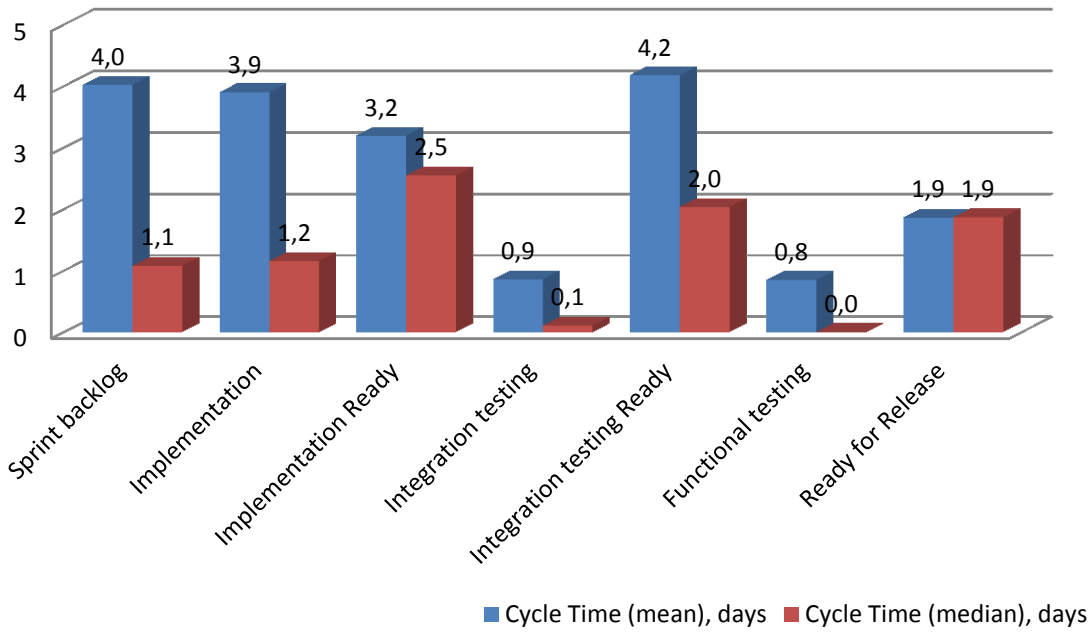

Figure 16: Team 3 mean cycle time in the development process in days

Team 3 has the most even development flow, meaning that there are no long waiting times in the backlog or "ready"-phases. The development flow is illustrated in Figure 16. Implementation takes roughly four working days (mean) and a bit over four days before the implementation is tested (Implementation Ready 3,2 days + Integration testing 0,9 days). Another four days are spent in queue after the integration testing. At this point, the team must wait for that the implementation is deployed in the testing environment and the acceptance testing can start. Functional testing referring to the acceptance testing, takes less than one working day. The following figure illustrates how much time is spent in a certain phase in percentages.

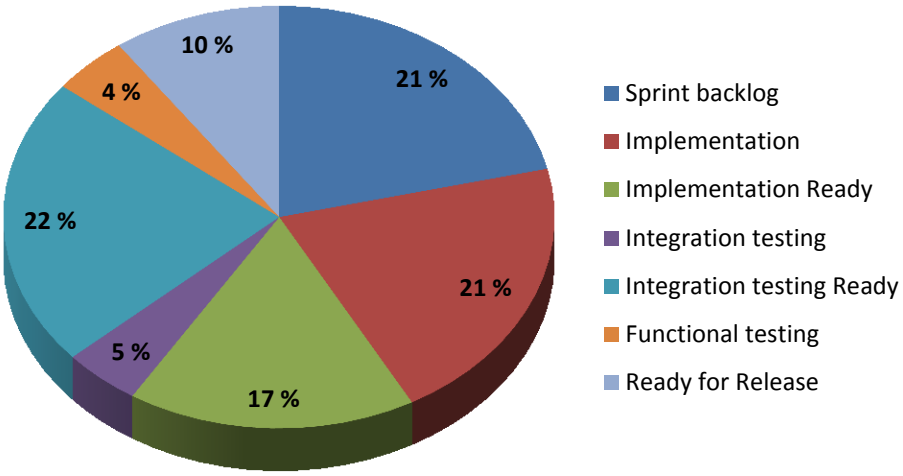

Figure 17: Team 3 mean cycle time per phase in percent

Figure 17 shows that roughly 70 percent of the total cycle time of the development process is spent in ”waiting”-phases (sprint backlog, implementation ready, integration testing ready, ready for release). The longest single queue time is between the Integration testing Ready and the Functional testing phases (Integration testing Ready, 22%). In practice, at this point a feature that is implemented and integration tested waits for the business representatives to start the acceptance testing (refers to Functional testing). Thus, 22 percent of the whole cycle time is spent on waiting for the acceptance testing.

5.4.4 Team 4 performance

The data represents the same information as with the previous two teams. Team 4 has divided its development process into eight phases. The development flow is illustrated in Figure 18.

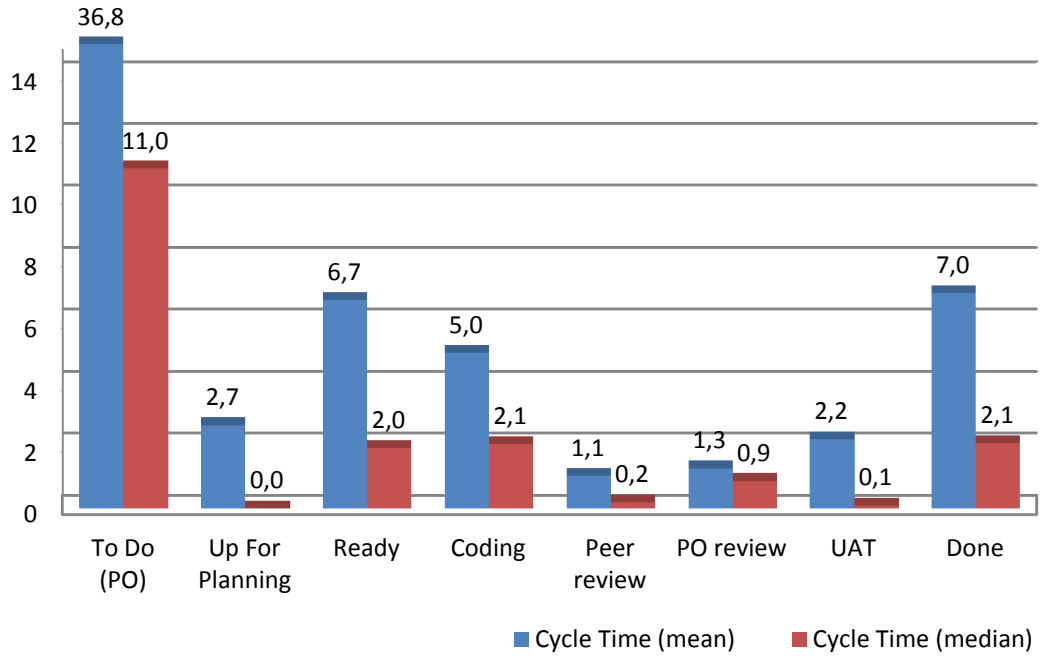

Figure 18: Team 4 mean cycle time in the development process in days

Up For Planning phase is the phase where the software features are further defined and processed before they are turned from an idea to a piece of software. Average time for planning the features is less than 3 days but the features wait for almost another 7 days before the team starts processing the feature. Coding, or implementation as other teams have defined, takes in average about 5 days. Peer review is part of coding but it is separated as its own phase. PO review is performed by the Product Owner in collaboration with a development team member. User acceptance testing (UAT) is performed in the user acceptance testing environment by the main user of the information system. Software features that are finished and waiting for a demo session are in the Done phase. After the implementation is presented to the customer, the software features are removed from the process.

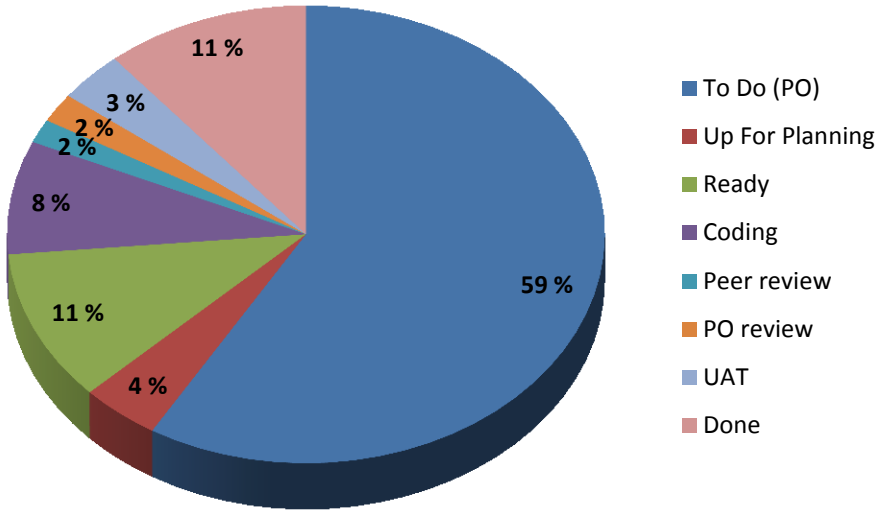

Figure 19: Team 4 mean cycle time per phase in percent

Figure 19 shows that roughly 81 percent of the total cycle time of the development process is spent in ”waiting”-phases (To Do (PO), Ready, Done). The To Do (PO) phase is taking 59% of the cycle time. It takes almost 37 working days before a development item is moved to the next phase in the process. This is because Team 4 does not have a separate “sprint backlog” phase which would shorten the cycle time of a single development item in one sprint. Team 4 is using pull-approach which means that the development team takes the tasks from the backlog once the previous one is finalized. Coding and peer review, considered as development team responsibilities, takes only 10 percent of the whole process.

5.4.5 Performance comparison

The previous sections presented the performance (cycle time) in each team studied in this thesis. This section collects the numerical metrics and we compare the metrics with each other. However, the comparison is difficult since processes differ in each team.

One possible way to compare the numerical metrics in this case is to divide the different development process phases into uniform process phases. In each team (except for Team 1), there are four common phases through which every individual requirement goes: backlog, the work done by the development team,

acceptance testing made by the customer, and the process in which the finished requirements are waiting for release. In Table 12, the numerical metrics are presented based on this division. Unfortunately, we cannot form the same metrics for Team 1 as for the other teams since there was not enough data collected for Team 1.

Team	Backlog	Development team	Acceptance testing	Release process	Total
Team 1	n/a	7,4	n/a	n/a	**n/a**
Team 2	23,2	7,3	5,9	20,9	**57,3**
Team 3	4,0	8,0	5,0	1,9	**18,9**
Team 4	38,1	15,5	2,2	7,0	**62,8**

Table 11: Mean cycle time divided by responsible parties

Backlog is the phase where the development tasks are waiting for the development teams' work. These tasks might need more work by the Product Owner before the development team can implement the task. The development team phase where the development team is implementing the development tasks contains steps such as coding, peer review, integration testing and so on. Those steps are in direct influence of the development team. Acceptance testing, on the other hand, requires resources from the business, who can actually make a statement whether the implementation fill the business need or not. Release process is again out of the control of the development team. The development tasks are ready to be released in this phase. To sum up, it is interesting to compare the development teamwork as well as acceptance testing work and how they differ over teams. These work phases are in direct control of the organization. However, we must remember that the flow efficiency of the whole development process will not be improved if only the work of the development team is improved.

As we can notice from table above, the numerical metrics indicate that the performance in each team is different. Perhaps the most valuable interpretation we can make here is that the teams are using their project management tools

differently and thus we are unable to compare the data without first unifying the usage of the tools. The processes in the project management tools should be unified, as well as the ways how the teams are using the tools e.g. when the team members move the development items in the tools from one phase to another.

6 Discussion

The purpose of this thesis was to examine the current ICT development process in the case organization, and to study whether the principles of the Kanban method are applicable for improving the ICT development process towards a more flow efficient process. The outcome of the literature review indicates that the Kanban method can improve process efficiency. As presented in chapter 3.3, adopting Kanban principles to the software development process decreased lead time significantly in multiple projects. Thus, one can assume that the potential exists in the case organization as well.

The improvement suggestions presented in this chapter are formed based on the knowledge gained during the study. First, the case organization should adopt Kanban principles on the project portfolio level (DLCM). Secondly, the organization should focus on implementing information system changes as a whole from the business need viewpoint, instead of implementing each change in a separate project. Finally, the organization should share the production responsibility between the Application supplier and the Production Service supplier because currently the deployment process is restricting the delivery flow.

6.1 DLCM with Kanban principles

The potential benefits of the Kanban method for improving flow efficiency in the software development processes were noticed while conducting the literature review for this thesis. Furthermore, the results of this study indicate that the organization is not working in an optimal manner. Currently, some development teams in the case organization are already following the principles of the Kanban method and they have improved their software development process. Even though each development team can improve their own performance, still the whole system is not flow efficient and the business needs are not fulfilled when required. Furthermore, the business cannot help customers creating value. In addition, some challenges, such as the internal resourcing policy, are in some level already recognized in the case organization but still the lack of understanding of the whole restricts making the necessary changes to the existing work practices. For these reasons, the same principles that the development teams are successfully using should be adopted on the DLCM level.

6.1.1 Visualize the process

The first step towards improved ICT development process in the case organization is to follow the first Kanban principle, hence to visualize the whole development life cycle model process. As both Kerzazi & Robillard (2013) and Maassen & Sonnevelt (2010) noticed, visualization helps in identifying bottlenecks in a development process. There are possibly multiple methods for process visualization. In software engineering, for example, Value Stream Mapping can be used for visualizing the development process (Staron & Meding, 2011). Process visualization should include all the major process phases.

Figure 20 represents one example of process visualization that could be used in the case organization. The process visualization in Figure 20 is based on the current DLCM process. The current DLCM process is visualized and the process phases are divided on a kanban board. The colorful sticky notes represent the development tasks that are flowing through the process. However, we must realize that in practice the DLCM process contains more steps within each process phase. For example, the Requirement analysis and preparation phase contains creating an architecture card, which is an individual step in that phase. If the organization wants to learn the processes inside DLCM phases such as the Requirement analysis and preparation phase, it can create a value stream map for that phase and identify the process.

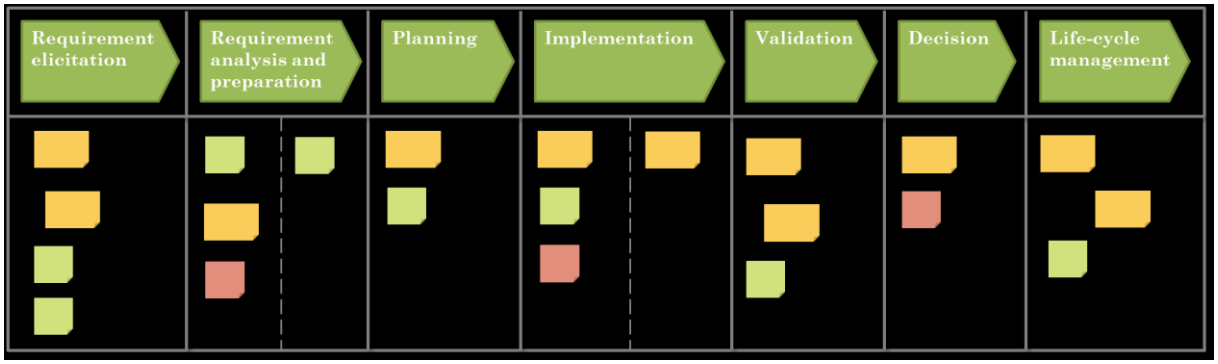

Figure 20: Visualized Development Life Cycle Model

In practice, the visualization of the DLCM process can be implemented either on a physical or on an electronic Kanban board. Greaves (2011) recommends using a physical kanban board since it can be easily modified in the early phases. The

visual kanban board should consist of all the identified process phases. Next, the existing development tasks are placed on the board in the corresponding phases, for example as sticky notes. Now that we have a visual board representing the development process, and the development tasks are presented on the board, we can notice the challenging phases of the process. As the theory explained, the challenging phases are those which include the most development tasks, thus are overloaded (Anderson, 2010). To create even more effective visualization, we can attach a number to the sticky note representing how many days each development task has been on the board. The number also helps in calculating lead time for each development task.

Visualization helps the organization understanding the issues that are present in the work flow (Anderson, 2010). In the case organization, visualization would point out what are the impacts while doing multiple development tasks simultaneously with the current resourcing policy. At the moment, there are more development tasks in the process than the organization can handle, as it was clearly showed in the Results section. A visualized development process will increase the understanding of the whole in the case organization.

Visualizing the process would inevitably highlight and possibly solve existing challenges in the case organization. However, starting to limit the work-in-progress can boost the positive impact of visualization. Thus, the next step towards a flow efficient ICT development process is to adopt the second Kanban principle: limit the WIP.

6.1.2 Limit the WIP

Now that we have visualized the development process, and we have some preliminary knowledge of the existing bottlenecks and inefficient process phases in the current development process, we need to start adjusting the development flow in order to improve the efficiency of the process. The Kanban method offers us a principle for that: limit the simultaneous work-in-progress (Anderson, 2010). We saw in chapter 3.1.4 how unfinished development tasks affect the throughput time of the whole system. If we cannot decrease the cycle time of the process significantly, the more development tasks there are in the process the longer the throughput time. Multiple case studies indicate that limiting the WIP clearly increases development efficiency (Middleton & Joyce, 2012), (Sjøberg, et al., 2012), (Shinkle, 2009), (Rutherford, et al., 2010), as we noticed in section 3.3. For

these reasons, we recommend that the organization start limiting the work-in-progress.

Next, we set a work-in-progress limit for each phase of the development process. The limit tells us how many development tasks a certain process phase can have simultaneously. A tentative work-in-progress limit guides the process in the beginning. However, the limits should not be static but rather reacting dynamically to changes in the process. In practice, a tentative WIP limit for the second phase of the DLCM should be adjusted according to the current resources. If we have five people that are processing the development tasks in the second phase of DLCM process, the WIP limit should be five. At the end of follow-up period, there were 29 development tasks in that phase which was inevitably far too many. Hence, the limit must be low enough to encourage finishing the actions in the current phase before pulling new task in to the phase. Figure 21 illustrates an example of work-in-progress limits in each phase of the DLCM process (Anderson, 2010).

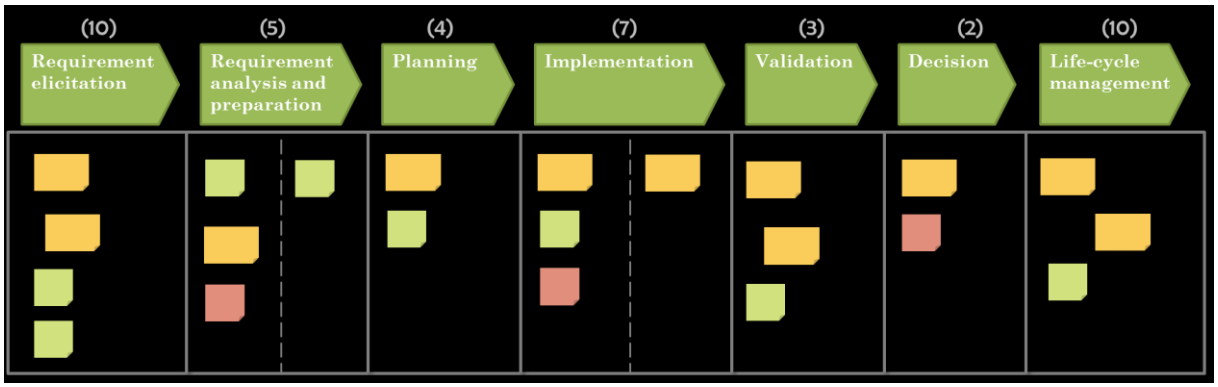

Figure 21: Development life-cycle model as a kanban board with WIP limits

Overall, the WIP limits should be adjusted according to the current available resources in the case organization. Resourcing policy was seen as one of the biggest challenges in the case organization. Now, as we limit the maximum amount of development tasks in each phase of the development process according to the available resources, the organization can truly see and understand that there are far too many development tasks in-progress simultaneously. Too many simultaneous development tasks are causing an inefficient ICT development process. The organization can start changing the resourcing policies, perhaps by sharing the knowledge required in one DLCM phase for a wider audience so that

the process does not rely on one person as in some DLCM process phases, e.g. system architect in the second phase of the DLCM. At the same time, the organization can understand the importance of finishing the development tasks all at once instead of pushing more and more into the process. Currently, the organization assumes that each development task will be ready in time even if the development process is overloaded.

6.1.3 Manage and measure the process

At least as important as visualizing the process and limiting the work-in-progress, is managing and measuring the development process. The third Kanban principle encourages us to manage and to measure the development process (Anderson, 2010). Processes most probably evolve as the work practices evolve, and therefore there is a need for continuous process improvement, as Nikitina, et al. (2012) noticed. The development process must be adjusted accordingly. However, the process should not be changed without being able to measure the effect of a change. Otherwise, how can we know if a change had a positive or a negative effect on the efficiency of a process?

For instance, the WIP limits, discussed in the previous chapter, are factors that change as the process evolves. The limits should be adjusted dynamically while the process evolves. We might notice that once a bottleneck is removed from one location, a new bottleneck appears in another location and the WIP limits should be adjusted accordingly. Measuring lead time for the development tasks is one possible way to measure the effects of a process change, as Taipale (2010) noticed. We must have a number before we adapt any changes into the process because otherwise we cannot measure the effects. The initial numbers for the development team performances were measured in this thesis. However, we still need to start measuring lead time for the whole ICT development process (DLCM) before we adapt any significant changes.

Once we have understood and visualized the ICT development process, we can measure the lead time for the whole process, from idea to production. In the DLCM, an idea is identified in the first phase of the process and deployed into the production environment in the fourth phase. We do not necessarily pursue to decrease the lead time while we change the process. Instead, we try to increase the efficiency of the process. However, at least the lead time should not increase while we implement changes into the ICT development process. Alternatively, we

can start measuring the cycle time of each process phase. The phases with the longest cycle time are the phases that should have the attention of the organization since those are the phases that are defining the efficiency of the whole ICT development process. Furthermore, if there is an unnecessary phase in the current development process, it must be removed from in order to increase the efficiency of the development process.

6.2 Business need centered development approach

As we have previously discussed in this thesis, the majority of the information systems in the case organization are highly dependent on one another. Other projects and dependencies were identified as bottlenecks in the current ICT development process during the interviews. Often a change in one information system causes changes in other information systems, especially in cases where the change concerns a whole business action across multiple information systems. At the moment, as discussed in chapter 2.3, development tasks that cannot be developed with a moderate effort or have dependencies to other information systems are carried out as projects. Often, a project is dedicated for the changes in one information system. If there is a need for changing something in another information system, it is implemented in another project. Each project has own its budget, schedule and scope. From the business need perspective, the current project based approach is not optimal, since each project is focusing on reaching own its goals, and not trying to fulfill the actual business need as a whole. We can see an illustration of the current project based development approach that is used in the case organization in Figure 22.

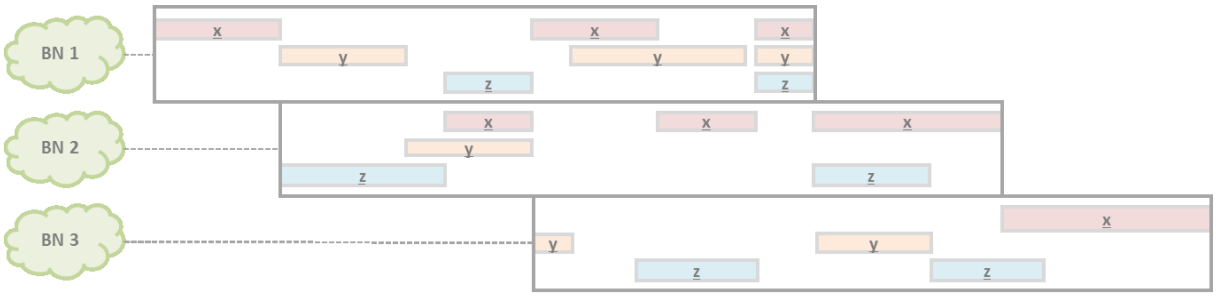

Figure 22: The existing project based development approach

The business does not receive the realization of the benefit from the changes until the whole business need is fully implemented in each required information

system. Additionally, as we develop changes as separate parts we create inventories of unfinished features. Inventories generate a need to manage the inventories with additional processes. At the same time, the lead time of each business need increases. In Figure 22, we can see how the lead time for a single business need increases as we complete the work as separate parts. The empty spaces between the projects (illustrated as squares in Figure 22) represent the time when the business needs are not processed. This extra time increases the lead time. If we consider for example the first business need (BN 1) in Figure 22, Project x starts to implement the BN 1 but they need to stop because they require something from projects y and z before they can continue their work. The same applies to the project z and project y. They always need something from someone else before they can continue their own part. Finally as the BN 1 is almost finished, there is a need to finalize the change in collaboration and the project resources are brought together.

For these reasons, there is a need in the case organization to start treating the development tasks and the implementation work from the viewpoint of the business needs. A business need centered development approach offers a simplified solution for the existing challenges in the case organization. The approach is designed to work on top of the Kanban principles discussed in section 6.1. The approach is presented in the Figure 23 below.

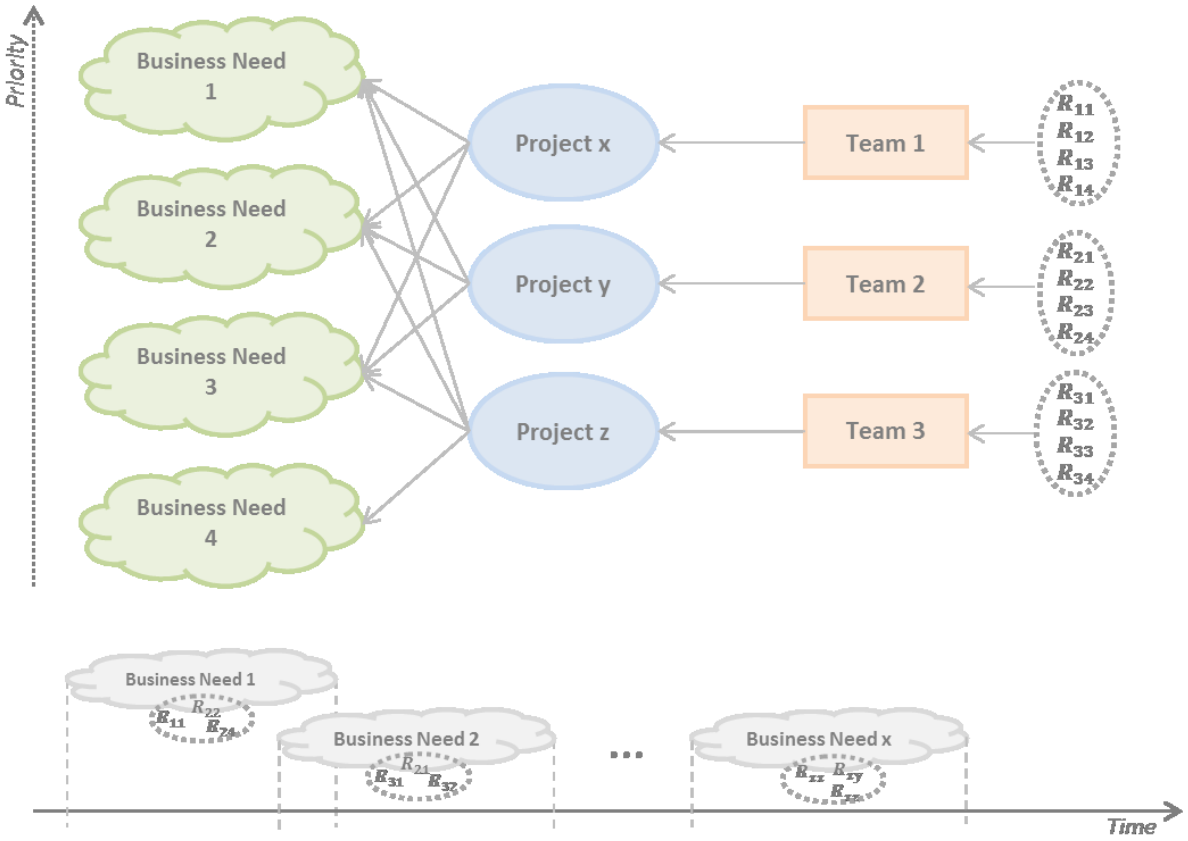

Figure 23: The business need centered development approach

The focus in the approach is on complete business needs (on the left in Figure 23), not projects as in the current DLCM. At the moment, a single business need can be developed in multiple projects, and implemented into multiple information systems, each separately. Instead of as individual projects, in the business need centered development approach the business needs are implemented as a whole in a single development team formed from the required resources. The business needs are implemented in a prioritized order following the Kanban principles. In other words, the business needs are pulled into the implementation (or any other phase of the process) once there is a slot (WIP limit) in the system. As illustrated in the Figure 23, often there is a team responsible for an information system or a set of information systems, and each team consists of resources. In this approach, all the necessary resources (developers and business advisers), possibly from different projects are collected into one physical location to work as one team for one goal. In practice, this "new" team has its own budget, schedule and scope for implementing the business need. Once the implementation work is finished and

the solution for the business need is in use and finally used by the business, the resources are again available for other work unless the same resources are allocated to implement the next business need that is a part of a wider strategic target (e.g. converting current services to e-service).

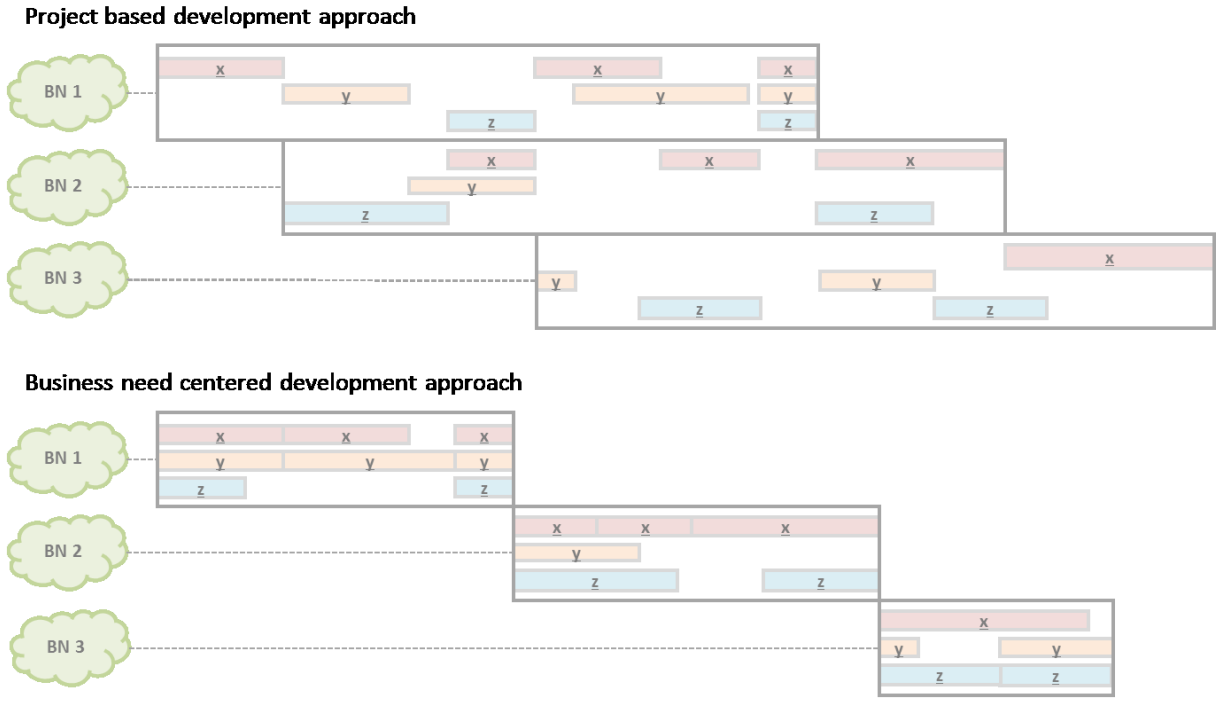

Figure 24: Comparison between project based and business need centered development approaches

The approach has many potential benefits in the case organization. First, the chaotic dependency jungle, as it is seen and treated in the case organization, will diminish significantly because the business needs are not completed as separate parts and the required resources are brought together. Currently, the organization is wasting huge amount of resources on dependency management even though the whole dependency jungle is a self-created issue. For the development teams, the dependencies are seen as bottlenecks that decrease the process efficiency. Secondly, lead time for a single business need from an idea to production will be much shorter than it is in the current project centered approach, as we can see in Figure 24. Actually in the example, all the three business needs are finished faster with the business need centered development approach. The business need centered development approach removes the

barriers between different projects and the waste that is generated with the current work practices. When applying the business need centered development approach the business needs are finalized at once, and there will not be individual parts of a business need waiting for other parts to be ready. If we study the manufacturing industry and the lean principles, we know that we should avoid inventories (unfinished products) as much as we can (Japan Management Association, 1989). Thirdly, once there are fewer dependencies and actions that are generated for managing the dependencies and unfinished business needs, the whole ICT development process will be more efficient. Also, the communication between the developers and other resources will increase since they all work in a shared space for the same goal. Additionally, there is no need to argue about budgets over different projects since the business need has the budget, not the individual projects. Finally, now that the teams have the necessary resources from the beginning, teams do not have to find time from the resources and thus the most significant bottleneck in the case organization is not a concern anymore.

Of course, implementing the business need centered development approach into the case organization also has challenges. First, the current budgeting policy requires projects meaning that the money is allocated for projects. Each change implemented in the information systems is allocated for a certain project. In the business need centered development approach the used money would be split possibly for multiple projects and at the moment there is no policy how the budget is distributed over projects. Alternatively, the whole "project budget" thinking could be changed to business need based budgeting, meaning that the budget is allocated for the business needs. However, we do not want to request a budget for each individual business need especially if the single business needs are only individual parts of some strategic targets. In that case the budget should be reserved for the whole strategic target such as "converting services into e-services". Secondly, the approach does not eliminate the challenge how the dependencies are identified. It is possible that we do not identify all the dependencies in the first place, and we cannot get the required resources in the middle of the business need work. Thirdly, the approach does not solve the existing prioritization challenge, thus, which business needs are more important than others. However, the approach offers a way for the directors to discuss about the right order in a simplified and visualized format. Finally, current projects have already made promises and engagements to implement certain changes, and thus the new approach would require reorganizing some of the promises and scheduling them again.

6.3 Shared production responsibility

Following the Kanban principles in the ICT development projects can possibly eliminate most of the challenges in the case organization. However, there is still one challenge in the case organization that will not be solved alone by following the Kanban principles, the challenges in the deployment process. Nevertheless, Kanban principles will visually show us the bottlenecks. Shared production responsibility improvement refers to the bottleneck that was partly introduced in chapter 2.6 and further discussed in chapter 5.1.2. Kanban principles would definitely solve the challenges that the current deployment process and support functions are causing if the current contracts with the production supplier did not disallow us from working physically in a closer collaboration.

Shared production responsibility simply means that the Application supplier and the Production Service supplier share the responsibility of the production environment. At the moment, the Production Service supplier does not allow the Application supplier to install the new versions of the software to the acceptance testing or the production environment. This policy causes a bottleneck and decreases the flow efficiency of the ICT development process, as stated in chapter 2.6. The improvement here would be to change the current policy, for example by first sharing the production responsibility of one information system and allowing the Application supplier to install a new version of software into the production environment. If the new arrangement shows us positive experiences for example as decreased deployment time or as fewer errors during the deployment process, the same policy could be used in other information systems as well.

6.4 Limitations of the study

The study was conducted in one type of development environment. Thus, the results represent challenges in a certain type of development environment, not necessarily generic software development process challenges. There were only four teams that were studied in this thesis. This fact also limits the usefulness of the results in a wider scope.

The amount of interviews in this study was small, and the interviewees represented only one type of a development environment. Ideally, there would have been more interviews from different development environments, both from

private and public sectors. Also, it would be useful to include those teams in the study which have common working practices and project management tools, thus the data would be more comparable with each other.

A limitation concerning the research method was that the interviews were not recorded and transcribed. It is possible that the interviewer made personal interpretation during the interviews because the environment and the challenges were already known in the organization. The project management tool study also had limiting factors. For example, the teams use their project management tools differently and the data is not necessarily reliable and comparable.

6.5 Future research

In the case organization, the next step is to adopt the Kanban principles and to take in to use the Business need centered development approach in the current work environment. The new approach could be used in a pilot "project" and if the results of the pilot are promising, the approach could be used more widely in the case organization. The efficiency of the entire development flow should be measured before and after the changes have been implemented so that we have data in which direction we are heading. Future research in the case organization should include a larger range of different development teams. In addition, both the whole project level efficiency and team level efficiency would be interesting topics for the future research.

This thesis only scratched the surface of what kind of possibilities the Kanban principles has. Future research in the field of software engineering could include studying the possibilities of the Kanban method in other development environments and learning how Kanban can solve flow efficiency challenges in other development projects outside the case organization. There is a need for further research in this area in order to answer the original research question in a wider scope. Additionally, it would be interesting to study whether the business need centered development approach is already used in other environments and what kind of benefits it brings.

7 Conclusion

The main purpose of this thesis was to address four research questions about the ICT development process in the case organization and the Kanban method.

> RQ1. What is the current ICT development process in the case organization?

We were able to identify the current ICT development process in the case organization and thus answer the first research question of this thesis. The ICT development process in the case organization is described in chapter 2.2. The current ICT development process can be examined from different perspectives. On one hand, if we examine the development teams and their processes individually, we might even notice that the teams are actually working quite efficiently. On the other hand, if we examine the whole ICT development process on the organizational level, we will notice much more sources of inefficiency.

> RQ2. How can the Kanban method help organizations to solve flow efficiency issues?

For the second research question, a literature review was conducted. The literature review indicated that adopting the Kanban method can have a significant impact on flow efficiency in software development processes. Studies support the claim that the Kanban principles increase the efficiency of software development process and shorten lead times. Visualizing the process and limiting the work-in-progress were seen as useful principles for identifying the challenges in the development process and for increasing the efficiency.

> RQ3. How can the case organization achieve a more flow efficient ICT development process with the principles of the Kanban method?

To answer the third research question, three improvement suggestions were formed based on the literature review and the findings concerning the current ICT development process in the case organization. First, the organization should adopt the Kanban principles in the project portfolio level as has already been started doing in the development teams. We have seen that Kanban principles increase the efficiency of the process in the development teams. However, every

project and development task should be treated in a common manner. Once the whole ICT development process is visualized, the organization gets a clear picture of the current status of the ICT development. Additionally, visualization highlights the process bottlenecks and thus helps the organization to start solving the issues.

Secondly, Business need centered development approach was suggested because at the moment, many development tasks are implemented in separate projects with different budgets, schedules, and scopes. The current project based approach often leads to a situation where the business needs are not satisfied as a whole and the business do not benefit from the individual software changes on time. Business need centered development approach encourages an organization to finalize the business needs all at once instead of splitting them into separate projects. However, there are several challenges that need to be eliminated before the new approach could be used in the organization.

Finally, the organization should work in closer collaboration with the Production Service supplier. Currently, the deployment process is a clear bottleneck in the ICT development process. The bottleneck exists because there are old contracts between the case organization and Production Service supplier. These contracts are causing unnecessary bureaucracy in the process because the responsibilities are strictly defined. In the future, the production responsibility should be shared between the Application and Production Service supplier. The case organization is only the enabler in this deal.

> RQ4. How did Kanban adoption help case organization with flow efficiency issues?

The fourth research question was partly answered in this thesis. Principles of the Kanban method were used in one development team. However, in this thesis it was not possible to examine how the improvement suggestions influenced the whole organization and the ICT development process after implementing the changes within this thesis. This is a subject for the future research.

References

Ahmad, M. O., Markkula, J. & Ovio, M., 2013. *Kanban in software development: A systematic literature review*, IEEE, pp. 9-16.

Anderson, D., 2010. *Kanban - Successful Evolotionary Change for Your Technology Business.* Washington: Blue Hole Press.

Anderson, D., Concas, G., Lunesu, M. I. & Marchesi, M., 2011. *Studying Lean-Kanban Approach Using Software Process Simulation.* Madrid.

Birkeland, J. O., 2010. *From a timebox tangle to a more flexible flow.*

Dybå, T. & Dingsøyr, T., 2008. Empirical studies of agile software development: A systematic review. *Information and Software Technology,* 50(9-10), pp. 833 - 859.

Fernandes, C., 2012. *There and Back Again: From Iterative to Flow ... and Back to Iterative!*, pp. 103-110.

Greaves, K., 2011. *Taming the Customer Support Queue: A Kanban Experience Report*, pp. 154-160.

Ikonen, M., 2010. *Leadership in Kanban software development projects: A quasi-controlled experiment.*

Ikonen, M., Kettunen, P., Oza, N. & Abrahamsson, P., 2010. *Exploring the sources of waste in Kanban software development projects*, pp. 376-381.

Ikonen, M. et al., 2011. *On the impact of Kanban on software project work: An empirical case study investigation*, pp. 305-314.

Japan Management Association, 1989. *Kanban - Just-In Time at Toyota.* New York: Productivity Press.

Kerzazi, N. & Robillard, P. N., 2013. *Kanbanize the release engineering process*, pp. 9-12.

Ladas, C., 2009. *Scrumban - Essays on Kanban Systems for Lean Software Development*. Modus Cooperandi Press.

Maassen, O. & Sonnevelt, J., 2010. *Kanban at an insurance company (are you sure?)*.

Middleton, P. & Joyce, D., 2012. Lean software management: BBC worldwide case study. *IEEE Transactions on Engineering Management,* 59(1), pp. 20-32.

Modig, N. & Åhlström, P., 2012. *This Is Lean*. First edition ed. Halmstad: Rheologica Publishing.

Neely, S. & Stolt, S., 2013. *Continuous delivery? Easy! Just change everything (well, maybe it is not that easy),* pp. 121-128.

Nikitina, N., Kajko-Mattsson, M. & Strale, M., 2012. *From scrum to scrumban: A case study of a process transition,* pp. 140-149.

Ohno, T., 1988. *Toyota Production System - Beyond Large-Ccale Production*. First Edition ed. Portland, Oregon: Productivity Press.

Oza, N., Fagerholm, F. & Munch, J., 2013. *How does Kanban impact communication and collaboration in software engineering teams?*, pp. 125-128.

Petersen, K. & Wohlin, C., 2009. A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *The Journal of Systems and Software,* 82(9), pp. 1479 - 1490.

Royce, W. W., 1987. *Managing the Development of Large Software Systems: Concepts and Techniques*. Montrey, California, USA, IEEE Computer Society Press, pp. 328 - 338.

Rutherford, K., Shannon, P., Judson, C. & Kidd, N., 2010. *From chaos to kanban, via Scrum*.

Schwaber, K. & Sutherland, J., 2013. *The Scrum Guide*.

Seikola, M., Loisa, H. M. & Jagos, A., 2011. *Kanban implementation in a telecom product maintenance*, pp. 321-329.

Shinkle, C. M., 2009. *Applying the dreyfus model of skill acquisition to the adoption of kanban systems at software engineering professionals (SEP)*, pp. 186-191.

Sjøberg, D. I. K., Johnsen, A. & Solberg, J., 2012. Quantifying the effect of using Kanban versus scrum: A case study. *IEEE Software,* 29(5), pp. 47-53.

Staron, M. & Meding, W., 2011. Monitoring bottlenecks in agile and lean software development projects - A method and its industrial use. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),* Volume 6759 LNCS, pp. 3-16.

Taipale, M., 2010. *Huitale - A story of a Finnish lean startup*.

Trafi, 2014. *Trafi Finnish Transport Safety Agency*. [Online] Available at: http://www.trafi.fi/en [Accessed 10 June 2014].

Wang, X., Conboy, K. & Cawley, O., 2012. "Leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems and Software,* 85(6), pp. 1287-1299.

Yin, R. K., 2003. *Case Study Research: Design and Methods*. Fifth edition ed. California: SAGE Publications.

Appendix A: Interviews

[I1] Head of KARKKI team, Owner of DLCM, development director, 08.05.2014

[I2] Head of Project Management Office (PMO), Owner of APMM, development director, 13.05.2014

[I3] Product Owner, Team 1, development director, 27.05.2014

[I4] ICT Product Owner, Team 3, special advisor, 03.06.2014

[I5] Scrum Master, Team 3, consultant, 04.06.2014

[I6] ICT Product Owner, Team 2, advisor, 06.06.2014

[I7] Scrum Master / Developer, Team 2, consultant, 10.06.2014

[I8] Scrum Master / Proxy-Product Owner, Team 4, consultant, 24.06.2014

Appendix B: Interview structure

Interview structure

Introduction

- Introducing the thesis and the topic
- Providing examples of resource efficiency and flow efficiency

Team overview

1. Which are the main responsibilities of the team?
2. What is your own role in the team?

Bottlenecks

1. Which type of bottlenecks you have noticed in the development process?
2. Which phases of the development process are bottlenecks?
3. Why there are such bottlenecks in the development process? What causes the bottlenecks?
4. Is there a way to eliminate the existing bottlenecks?

Variation

1. In what way the development tasks vary in the development process?
2. What causes variation in the development tasks?
3. Which kind of effects the variation causes for the development process?
4. Could the development tasks be unified in order to avoid variation?

Work-in-progress

1. Is the team processing multiple tasks simultaneously?
2. How many development tasks there are currently in progress?

Further discussion