UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

COMPUTER SCIENCE

Kim Nordkvist

REQUIREMENTS DISCOVERY FOR A PRODUCTION MANAGEMENT SOFTWARE

Master's thesis for the degree of Master of Science in Technology submitted for inspection, Vaasa, 14.5.2013.

Supervisor Professor Jouni Lampinen

Instructor M.Sc. (eng.) Tommi Soini

TABLE OF CONTENTS **page**

ABSTRACT iii
TIIVISTELMÄ iv
LIST OF FIGURES v
LIST OF TABLES vi

1. INTRODUCTION 1
   1.1. Background and motivation 1
   1.2. Objectives of the thesis 2
   1.3. Focus, scope and limitation 2
   1.4. Outline of the thesis 3
2. CONDITION BASED MAINTENANCE 4
   2.1. Condition monitoring services 4
   2.2. Service Agreements 5
   2.3. CBM production management 6
3. REQUIREMENTS ENGINEERING 7
   3.1. Software process 7
   3.2. Software process model 8
      3.2.1. The waterfall model 9
      3.2.2. Wärtsilä project model 10
   3.3. Software requirements 13
      3.3.1. Business requirements 14
      3.3.2. User requirements 14
      3.3.3. System requirements 14
      3.3.4. Functional and non-functional requirements 15
      3.3.5. Domain requirements 15
   3.4. Requirements engineering process 16
      3.4.1. Feasibility study 17
      3.4.2. Requirements elicitation and analysis 19
      3.4.3. Requirements specification 21
      3.4.4. Requirements validation 24
      3.4.5. Requirements management 25
   3.5. Unified Modelling Language 25
   3.6. Use case approach to requirements gathering 26

3.7. Use cases 27
3.7.1. Scenarios 27
3.7.2. The actor 28
3.7.3. The use case diagram 28
3.7.4. Documenting use cases 30
3.7.5. Writing use cases 31
3.8. Use cases as tools of requirements specification 36
3.9. An approach to translate user needs into user requirements 37
3.10. Challenges and issues with requirements gathering 40
4. THE CURRENT STATE OF CBM PRODUCTION 43
4.1. Background 43
4.2. Stakeholders 44
4.3. CBM delivery process 45
4.4. Tools used today 46
4.5. Identified problems 47
4.6. Why is a new tool needed? 49
5. THE REQUIREMENTS DISCOVERY 51
5.1. Scope 51
5.2. User classes 53
5.3. Requirements elicitation technique 54
5.4. Software Requirements Specification 54
5.5. Functional requirements 56
5.6. External interface requirements 66
5.7. Other non-functional requirements 67
6. DISCUSSION AND CONCLUSION 69

REFERENCES 71

UNIVERSITY OF VAASA
Faculty of technology
Author: Kim Nordkvist
Topic of the Thesis: Requirements Discovery for a Production Management Software
Supervisor: Prof. Jouni Lampinen
Instructor: M.Sc. (eng.) Tommi Soini
Degree: Master of Science in Technology
Degree Programme: Degree Programme in Information Technology
Major of Subject: Software Engineering
Year of Entering the University: 2002
Year of Completing the Master's Thesis: 2013 **Pages:** 73

ABSTRACT:
Due to the expanding Condition Based Maintenance (CBM) business at Wärtsilä Oyj a software tool was needed to ease and secure the maintainability of the installed base, delivery requirements, contract and invoicing information. Another aim with this tool was to achieve a degree of predictability of the CBM services by having information available about forthcoming requests for CBM systems and services. As a result, a tool with up-to-date information would serve many stakeholders with all available CBM related information but also remarkably reduce the work load needed to maintain this information manually. The objective of this thesis was to gather and document the requirements.

For the requirements discovery a use case based requirements elicitation technique was selected. It was selected because use cases can be documented in a structured way and also because they are a good tool to communicate the behavioural functions of a system between users and software developers. Requirements were also discovered by studying documents, tools, and process, but also by arranging interviews and having discussions with the stakeholders. All the requirements were documented in a Software Requirements Specification using a template from IEEE Std 830-1998.

The result of this thesis is a Software Requirements Specification that defines the requirements for this new tool. It was observed that even though it takes only a few minutes to learn read use cases, learning to write good use cases requires much more effort. In addition, the use cases are an important input when making the functional design specification, but they also serve as a base for designing the test cases that the new software will have to meet. Finally, the most important lesson learned was the importance of specifying software requirements that cannot be ignored.

KEYWORDS: user requirements, requirements discovery, use cases

VAASAN YLIOPISTO
Teknillinen tiedekunta
Tekijä: Kim Nordkvist
Diplomityön aihe: Tuotannon hallinnan ohjelmiston vaatimusten määrittely
Valvojan nimi: Prof. Jouni Lampinen
Ohjaajan nimi: DI Tommi Soini
Tutkinto: Diplomi-insinööri
Koulutusohjelma: Tietotekniikan koulutusohjelma
Suunta: Ohjelmistotekniikka
Opintojen aloitusvuosi: 2002
Diplomityön valmistumisvuosi: 2013 **Sivumäärä:** 73

TIIVISTELMÄ:
Wärtsilän Condition Based Maintenance (CBM) liiketoiminta on kasvanut vuodesta toiseen, ja sen takia ilmeni tarve ohjelmistolle, joka turvaisi ja helpottaisi toimitusten, sopimusten ja laskutuksen tietojen ylläpitoa. Toinen tavoite työlle oli myös mahdollistaa uusien toimitusten ennustettavuus. Tämän seurauksena uusi ohjelmisto palvelisi monia sidosryhmiä ajan tasaisilla tiedoilla, mutta se myös vähentäisi merkittävästi käsin tehtävää tietojen syöttöä ja ylläpitoa. Tämän työn tavoite oli kerätä ja kirjata uuden ohjelman vaatimukset.

Vaatimusten havaitsemiseksi käytettiin käyttötapauksiin perustuvaa tekniikkaa. Tällä havaitsemistavalla oli monia hyötyjä. Ensinnäkin käyttötapaukset voidaan jäsentää hyvin ja niitä on myös helppo lukea ja tulkita ja siten myös kommunikoida vaatimukset tuleville käyttäjille ja kehittäjille. Vaatimusten havaitsemista suoritettiin myös tekemällä haastatteluja ja käymällä keskusteluja sidosryhmien kanssa, mutta myös tutkimalla nykyisiä dokumentteja, sovelluksia ja prosesseja. Kaikki vaatimukset kirjattiin ohjelmiston vaatimusmäärittelyyn, joka perustui IEEE 830-1998 standardiin.

Tämän työn tulos on vaatimusmäärittely uuden ohjelmiston vaatimuksista. Yksi havainto työstä oli se, että käyttötapauksia on helppo oppia lukemaan, mutta hyvän käyttötapauksen kirjoittaminen vaatii taitoa ja aikaa. Lisäksi käyttötapaukset ovat tärkeä syöte seuraavaan suunnitteluvaiheeseen, mutta myös testitapauksien suunnitteluun. Lopuksi kaikkein tärkein havainto oli ymmärtää, kuinka tärkeää on suorittaa vaatimusten määrittely, jota ei voi jättää huomioimatta.

AVAINSANAT: käyttäjien vaatimukset, vaatimusten kartoittaminen, käyttötapaukset

LIST OF FIGURES **page**

Figure 1. The concept of CBM services (Wärtsilä 2012b). 5
Figure 2. The software life cycle (Sommerville 2007: 66). 9
Figure 3. Operational development project lifecycle phases. (Wärtsilä 2007: 6) 11
Figure 4. The requirements engineering process. (Sommerville 2007: 143) 16
Figure 5. The requirements engineering domain (Wiegers 2003: 13). 17
Figure 6. The requirements elicitation and analysis process (Sommerville 2007: 147). 20
Figure 7. IEEE Std 830-1998 (1998) proposed table of contents for a SRS document. 22
Figure 8. Use case diagram (Rumbaugh et al. 2004: 78). 29
Figure 9. Use case relationships (Rumbaugh et al. 2004: 80). 30
Figure 10. Use case levels (Cockburn 2000: 62). 33
Figure 11. Task sequence diagram (Kujala et al. 2001: 47). 37
Figure 12. Stakeholders and interactions. 44
Figure 13. The CBM delivery process. 45
Figure 14. Context diagram of the production management software. 51
Figure 15. Table of contents for the SRS. 55
Figure 16. Context level use case diagram. 60

LIST OF TABLES **page**

Table 1. A shortened sample Actor-Goal List (Cockburn 2000: 37). 32
Table 2. A shortened sample of use case briefs (Cockburn 2000: 38). 32
Table 3. Top-ten problems and pitfalls when writing use cases (Lilly 1999). 34
Table 4. User need table (Kujala et al. 2001: 48). 38
Table 5. A use case written based on the user need table and a list-based requirements document (Kujala et al. 2001: 48). 39
Table 6. Listing of identified problems in CBM production. 47
Table 7. In/out list of features for the production management software. 52
Table 8. User classes of the production management software. 53
Table 9. Actor profile table 57
Table 10. Actor-goal list for the production management software. 58
Table 11. Use case: Upload CBM report 62
Table 12. Use case: Distribute CBM report. 63

1. INTRODUCTION

To ensure the success and growth of operations a business always need to be alert to changes and avoid the corner of remaining stuck with old routines. Everything change, some things more often than others, but this always makes it more important to take actions. Further, we cannot avoid the increase in need for information. Information is everything, but if it is not up-to-date and valid it can be worthless. These all are reasons that urge us to develop new tools, new software to survive and be successful.

1.1. Background and motivation

In year 2001 a Condition Based Maintenance (CBM) centre was established at Wärtsilä Oyj in Vaasa. The purpose of it was to offer condition monitoring services to the customer. The delivered engines were connected to the CBM centre so that the operation data could be analysed and then the condition of the engine reported to the customer. This was also internally an interesting opportunity to start follow-up how new technology behaved out on the field. Today, twelve years later more than 400 installations and 2000 engines worldwide are connected to the CBM centre and the amount of connected engines is estimated to continue increasing. In addition, the CBM concept has been expanded from the 4-stroke engines in the beginning to also cover other portfolios such as 2-stroke engines and propulsion equipment. As the number of connected installations has increased but the working procedures remained mostly the same as ten years ago, the work needed to maintain up-to-date information about connected installations and engines has become more difficult and challenging.

Today, CBM is normally a part of long term service agreements, which involve additional information to be managed about contracts and invoicing. In addition, another important application for CBM is to monitor installations under warranty. The information that is required for an effective CBM production is scattered around at different stakeholders and saved in different applications. This causes a lot of work to request and obtain the information from the involved stakeholders that is needed for the CBM production. Further, when the availability of information is poor, also the visibility into the future concerning coming CBM service requests is non-existent, which again will result in difficult resource planning.

Consequences of all the mentioned information deficits are delays in deliveries and invoicing, and also a low forecasting possibility. Also, the cost transfers of activities to orders will be unplanned and difficult. Further, all this will result in reduced customer satisfaction and lost business opportunities, and also an increased need of resources to manage customer, contract and invoicing information. Therefore, a project, where I acted as the project manager, was started to develop a tool for managing production of the CBM services.

1.2. Objectives of the thesis

The objective for the project was to develop a production management tool that would remarkably reduce manual work required to maintain CBM system and services delivery information. The required information would be up-to-date and in one place. Further, it would enable efficient invoicing and cost transfer of activities on schedule. Finally, it would improve forecasting and resource planning. As the scope of this project was quite extensive and it would also take a considerable amount of time, only a part of it was to be included in this thesis work.

The main objectives of this thesis work were to elicit, analyse, and specify the requirements for the production management tool.

1.3. Focus, scope and limitation

The focus was set on collecting the requirements and specifying the requirements. This included the work to identify the stakeholders as well as selecting elicitation technique. Further, all the requirements needed to be analysed, and specified into a software requirements document. The most focus was set on specifying the functional requirements, and hence some tasks was left out-of-scope of this thesis. These tasks were to prioritize the requirements, making a traceability matrix as well as a dependency matrix.

The Wärtsilä Project Management Office (WPMO) offers a wide range of directives and guidelines about how an operational development project should be executed. They also

offer templates for many of the different documents that are produced by projects. These templates regarding how to document the requirements could have limited this work. But, in the end the work for this thesis was carried out outside of the official project, and hence this work was not limited by the WPMO directives and templates.

1.4. Outline of the thesis

The thesis is organized as follows. Section 2 introduces the CBM concept and gives a definition on production management. Section 3 gives an overview of the literature concerning the software development process and different process models, requirements engineering, and use cases. Section 4 describes the current state of CBM production management practises, and also motivates the development of a tool. Section 5 presents the result from the requirements discovery. Finally, in section 6 the result is discussed and the conclusions made.

2. CONDITION BASED MAINTENANCE

Wärtsilä is a global company that provides complete lifecycle power solutions for both marine and power plant applications. The emphasis is on the technological and total efficiency to maximize environmental and economic aspects for its customers. The main divisions of the company are Ship Power, Power Plants, and Services. Ship Power provides solutions that include products, systems, and services for the marine industry. The products, ships machinery, propulsion, and manoeuvring systems cover all types of vessels and offshore applications. Power Plants is a leading supplier of modern, highly efficient, and dynamic power plants for a decentralized power generation market. The Power Plants offer the customers flexible capacity in both urban areas, but also in the most demanding distant environments. Services support its customers throughout the lifecycle of their installation. With a large network and a comprehensive portfolio of services, Services provide service, maintenance, and reconditioning solutions for both marine and power plant applications. In addition, Services has also launched new innovative services like the predictive and Condition Based Maintenance services. (Wärtsilä 2012a)

Wärtsilä has approximately 18000 employees and operations in 70 countries. The net sales for 2011 were EUR 4.2 billion. Wärtsilä is listed on the NASDAQ OMX Helsinki stock exchange in Finland. (Wärtsilä 2012a)

2.1. Condition monitoring services

Condition Based Maintenance (CBM) is a service that the Wärtsilä Services division introduced in year 2001, when a CBM centre was established in Vaasa. The basic concept of the CBM service is to collect information from sensors on the engine to determine its condition and to prevent failures, see figure 1. First, the raw sensor measurements are transferred over the Internet to a central database, in which it is stored. Then, a reference performance analysis tool analyses the raw data to find deviations and abnormalities. Continuous follow up is made by engine experts who also make further analysis concerning the engine condition. The experts also write monthly reports for each installation, where they comment on the engine condition and give

recommendations for preventive maintenance. (Vägar 2012; Wärtsilä 2009; Wärtsilä 2012b)

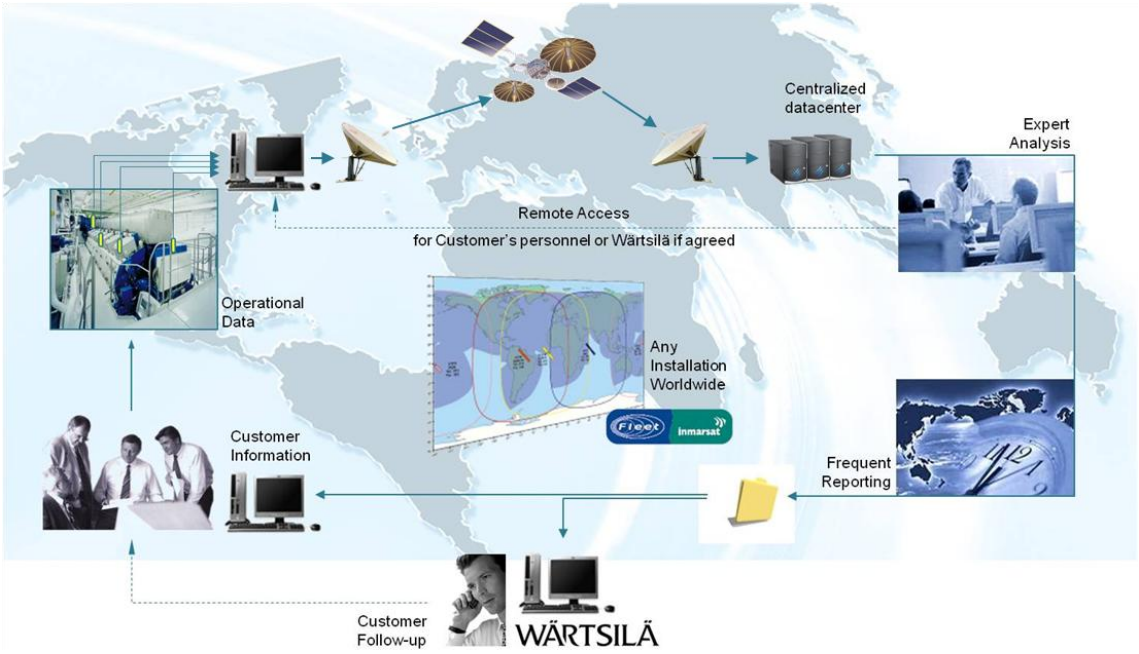

Figure 1. The concept of CBM services (Wärtsilä 2012b).

The CBM service gives the customers advantages such as cost savings by increased equipment availability and fuel efficiency. Also, the performance targets of the engine can be achieved by fine tuning the operation parameters. The CBM customers range from all types of marine application like cruise ferries, tankers and offshore applications to power plant customers.

Today more than 400 installations and 2000 engines are connected and monitored from this centre and numbers are steadily increasing. (Vägar 2012; Wärtsilä 2009)

2.2. Service Agreements

Today, CBM as service is an important part of different types of long term service agreements, and hence CBM is not anymore sold as a standalone product. The different

agreement types range from concise agreements that include dynamic maintenance planning, risk evaluation, training, and planning support to agreements that include complete asset management. A Service Agreement gives added value by guaranteeing to optimize the lifecycle efficiency of the customer's investment. (Wärtsilä 2012c)

2.3. CBM production management

The BusinessDictionary.com (2012) defines production management as:

> "The job of coordinating and controlling the activities required to make a product, typically involving effective control of scheduling, cost, performance, quality, and waste requirements." (BusinessDictionary.com 2012)

CBM production management is about managing delivery of the CBM system but also to manage the delivery of CBM services. The CBM system delivery can be considered as a project that occurs only once in every lifecycle of the CBM product, while the CBM services delivery is continuous work to deliver the service. The delivery of CBM services continues until an agreement ends or the service in not needed any more. Then the site is disconnected. The more specific activities will be discussed in section 4, as a part of the current state of CBM production management. (Vägar 2012)

3. REQUIREMENTS ENGINEERING

The term requirements engineering has been invented to cover all the activities that involve discovering, documenting, and maintaining requirements for computer-based systems (Sommerville & Sawyer 1998: 5).

This chapter will first discuss the software process and software process models to give the reader a basic understanding of the software development domain. Then the different types of software requirements will be presented. Next, the requirements engineering process will be discussed including all the sub-processes of it. Also, the Unified Modelling Language (UML) will be briefly explained as well as a use case approach to requirements gathering. Next, the use cases will be described in detail and how these can be used as tools for the requirements specification. Finally, some challenges and issues regarding requirements gathering will be discussed.

3.1. Software process

Sommerville (2007: 64) defines the software process as a set of activities that leads to the production of a software product. He continues that this can be to create a software product from scratch or to extend or modify existing software. Further, he describes that the software process is a complex, intellectual, and creative process that is difficult to automate. One reason for this is the great diversity of software processes, though there is no ideal process and it is common that organizations use their own approach. Moreover, Ghezzi, Jazayeri and Mandrioli (2003: 385) add that the software production process is characterized by high instability. With this they mean that the products themselves must evolve because the requirements change continuously. Sommerville (2007: 64) also mention that the requirements for the process may vary, depending on the characteristics of the system that is being developed. Despite of the many variations of software processes, he lists some fundamental activities that are common to all of them:

1. *Software specification*. The work to define the functionality and constraints of the software to be developed.
2. *Software design and implementation*. The software is designed and programmed to meet the specifications.

3. *Software validation*. Validation of the software to ensure that it meets the customer requirements.
4. *Software evolution*. The software must be developed to meet changing customer requirements.

3.2. Software process model

Sommerville (2007: 8-9) defines that a software process model is a simplified description of a software process. This simplified description presents one point of view of a software process. Furthermore, he argues that most software process models are based on one of three general models or paradigms of software development. These are:

1. *The waterfall approach*. The above mentioned software process activities are represented as separate process phases such as: requirements specification, software design, implementation and testing. Each phase is completed and the development continued on the next phase.
2. *Iterative development*. The activities of specification, development and validation are overlapped in this approach. The idea here is to have an initial system made from very abstract specification. The development is then iterative; the initial system is refined until it satisfies the customer's needs.
3. *Component-based software engineering*. In this approach the system development process is to focus on integration of existing system parts instead of developing the parts from scratch.

Sommerville (2007: 65) also adds that these models are often used together, particularly for large systems development. He also mentions that in fact the Rational Unified Process combines elements of all these models.

Ghezzi et al. (2003: 388-390) state that software process models are important, because the concern for quality and awareness of production process importance has increased, but also while the aim with it is to improve time to market and reduce production costs. He continues that processes are also important, because experience has shown that these have critical influence on quality. If the processes can be controlled also the quality will be better. This is particularly related with software production. Ghezzi et al. (2003: 390) also emphasize the importance of a transparent production process scheme instead of a

black box type of process scheme. The transparency allows the customer to understand what is going on and lets them observe the products produced during the process.

Ghezzi et al. (2003: 388) conclude that the purpose of the software process model is: first, to guide the software engineers on the activities and in which order they should be carried out, and secondly, they give a framework for them to estimate resources, define intermediate milestones, and monitor progress, which are managing development and maintenance activities.

In the next sections the activities involved in the waterfall model and Wärtsilä project model will be presented in more detail.

3.2.1. The waterfall model

The waterfall model is the oldest model for software engineering and it is also sometimes called the classic life cycle (Pressman 2005: 47). Ghezzi et al. (2003: 6-8) describe that the phases in the waterfall model progress in an orderly and linear fashion and each of them has a well-defined starting and ending point. Further, they state that each phase also has clear identifiable deliverables to the next phase. Figure 2 depicts the waterfall model or also called the software life cycle according to Sommerville (2007: 66).

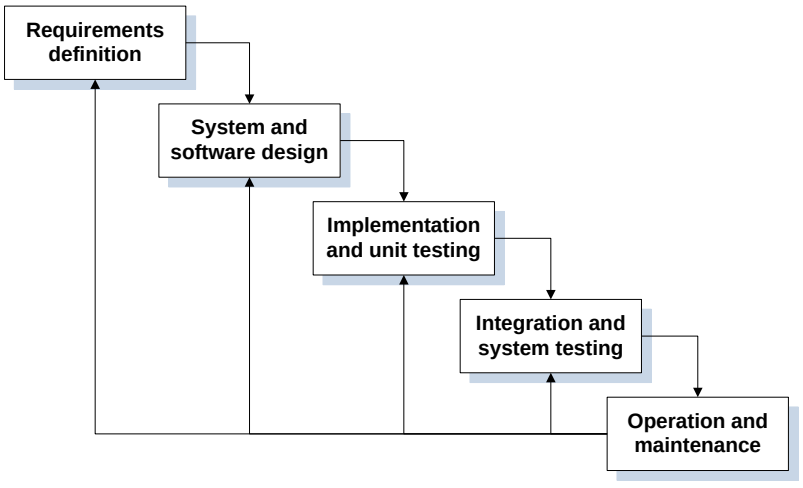

Figure 2. The software life cycle (Sommerville 2007: 66).

Sommerville (2007: 67) describes the fundamental development activities of the software life cycle as follows:

1. *Requirements definition*. By consulting the system users the system's services, constraints, and goals are identified. These are then documented in detail as a system specification.
2. *System and software design*. The system design establishes an overall system architecture, while the software design describes the detailed design and relationship of each software unit or module.
3. *Implementation and unit testing.* The software is programmed as a set of programs that are also tested and verified to meet the specifications.
4. *Integration and system testing.* The set of programs or units are integrated into a system. The system is then tested and verified that it meets the software requirements. Finally, the software is delivered to the customer.
5. *Operation and maintenance.* The software is put into use, errors are corrected, and it is also improved as new requirements emerge.

Haikala & Märijärvi (2004: 37) state that there are many variations of the waterfall model, but at least the above mentioned phases can typically be distinguished. They also add that the requirements definition phase is often preceded by a feasibility study or requirements study.

3.2.2. Wärtsilä project model

The Wärtsilä Project Management Guide (Wärtsilä 2011) is a document describing the Wärtsilä Project Model and it offers common and efficient project management practices that are aimed for all Wärtsilä employees working in projects. It covers the three main categories of projects: customer delivery projects, product and solution development projects, and operational development projects. The guide is written by the Wärtsilä Project Management Office (WPMO) and is based on the Project Management Institute's Project Management Body of Knowledge, PMBOK® Guide. The guide content and structure is based on the ABC Project Model™ developed by the Project Institute Finland Ltd. and further it is tailored to fulfil the specific needs for Wärtsilä. (Wärtsilä 2011: 2, 7)

The operational development project model, which is typically used for software projects, can be further categorized into three main types of projects; projects that aim to improve quality, processes or tools, capacity adjustments projects, and business development projects (Wärtsilä 2011: 7). This project model will next be presented in more detail.

The Wärtsilä Operational Development Project Guidelines (Wärtsilä 2007: 5) is a document that describes the main phases and activities of operational development projects. The main phases of the operational development project lifecycle are initiate, plan, execute, and close. Further, the execution phase is divided into the sub phases develop, validate, and deploy. In addition, the Wärtsilä Project Management Guide (Wärtsilä 2011: 8) adds two subsequent project related phases outside the project: the explore phase and the evaluate benefits phase. Figure 3 gives an overview of the project lifecycle phases in an operational development project. Each phase will be introduced in more detail later on in this section.

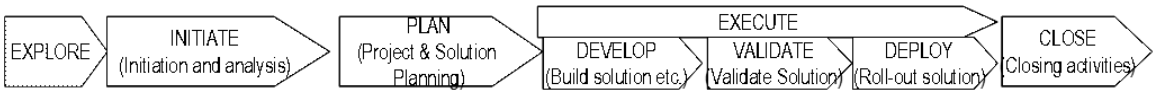

Figure 3. Operational development project lifecycle phases. (Wärtsilä 2007: 6)

Gates, which are mandatory decision-making points, separate each project lifecycle phase. At each Gate the achieved results are evaluated and the decision maker makes the decision whether the project is continued. Other possible outcomes from this decision could be: that the project is terminated, must be redefined and approved again later, or the project could be put on hold. The Wärtsilä Project Model defines five mandatory Gates (Wärtsilä 2011: 9):

- G0 “Start project”
- G1 “Start planning”
- G2 “Start execution”
- G3 “Start closing”
- G4 “Close project”

In the following sections the main pre-requisites, purpose, key activities, and results are briefly described for each phase in the project lifecycle.

The first phase, project initiate phase starts after that the Gate 0, start project, is approved. Other prerequisites for this phase are that a project proposal document including a detailed plan for initiate phase and also a rough plan for the entire project should have been approved. Further, a project owner and manager should have been appointed and the project approved for initiation. The purpose of this phase is to freeze the requirements and also to complete the preliminary project scope. The key activities of this phase are to: carry out project initiation according to the plan, setup project infrastructure, agree upon business requirements, make high-level overall project plan more accurate, make and approve the detailed plan for project plan phase. Finally, the result of the initiate phase should be a detailed plan on how to complete the next phase, the project plan phase. (Wärtsilä 2007: 6)

The prerequisites for the planning phase are that the project is approved for planning and the financial approvals for project planning are given. To freeze the detailed project scope and to plan how the project scope can be delivered are some of the purposes for this phase. In addition, an effective execution of the next phase should be ensured. The planning phase key activities are to perform project planning and freeze the project scope. Furthermore, to the key activities also count to define and approve solution design. This should be done on such a level that the project plan, business case, and potential vendor selection can be completed. Also, the project business case should be finalized. If a vendor is to be used it should also be selected during this phase. The detailed plan for project execution should also be made and approved. Finally, the planned solution should be checked that it is feasible. The outcome from the planning phase is a detailed requirements specification and a chosen solution as well as a detailed plan for the project execution phase. (Wärtsilä 2007: 6)

In operational development projects the execution phase is typically divided into the sub phases: develop, validate, and deploy. The main purpose of each sub phase is to perform the solution development, validation, and deployment according to plan. In more detail the key activities of the execution develop phase are to finalize the detailed solution design, build the solution, define test plans and protocols, and define and approve model for support and continuous development of the solution to be delivered. Deployment and support strategies should also be defined. During the validation phase the user

acceptance tests or a pilot should be performed in a way so that the complete solution can be validated. Also, the detailed deployment plan should be finalized and support and continuous development setup of the solution should be delivered. In the deployment sub phase, the activities are to deploy the solution and also to complete user training. Also all outstanding project issues should be identified and responsibilities and time-line for actions should be agreed on. Furthermore, the solution should get acceptance and be handed over to the support organization. (Wärtsilä 2007: 7)

The main purpose for the closing phase is to get the project closed by completing all the closing routines. This includes verifying that all necessary project activities has been completed and closed. Further, the outcome and lessons learned is an important fact that needs to be discussed. Some of the key activities are to: archive the project, collect and communicate lessons learned, settle business case follow-up procedures, and sign-off the project formally. Finally, all resources should be released and recommended further actions should be communicated to people responsible. (Wärtsilä 2007: 7)

3.3. Software requirements

According to Sommerville (2007: 118) the term *requirements* is not used in a consistent way. Specifically, he mentions that the level of detail of the requirement can vary from abstract statements to detailed definitions of system functions. Also, Wiegers (2003: 7) agree that the software industry lack common definitions for the term. Further, according to the SWEBOK® (2004: 37) the literature sometimes call the system requirements as user requirements. Hence, a clarification is needed. The IEEE Standard Glossary of Software Engineering Terminology (1990) defines a requirement as:

1. "A condition or capability needed by a user to solve a problem or achieve an objective."
2. "A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents."
3. "A documented representation of a condition or capability as in (1) or (2)."

Wiegers (2003: 8–9) complement this definition and consider a requirement as a property that a product must have to give value to a stakeholder. Further, he

distinguishes three levels of requirements: business requirements, user requirements, and functional requirements. He adds that every system also has an assortment of non-functional requirements.

3.3.1. Business requirements

Wiegers (2003: 9) explains that the business requirements, which are the highest level of requirements, state the objective of the organization or customer – stating why the system is needed and what the organization hopes to achieve. These are often stated in a vision and scope document or a project charter.

3.3.2. User requirements

Sommerville (2007: 118, 127) defines the user requirements as statements in natural language of the services that the system is expected to provide, but also the constraints under which it must operate. The user requirements describe the user goals or tasks that the user must be able to perform with the product in a way that is understandable by the user without detailed technical knowledge. Further, these descriptions of the requirements should avoid specifying system design, and instead focus on specifying external behaviour of the system. Wiegers (2003: 9) suggest that use cases, scenario descriptions, and event-response tables are valuable ways to represent user requirements.

3.3.3. System requirements

The SWEBOK® (2004: 37) defines that the system requirements are the requirements for the system as a whole, while Somerville (2007: 129–131) explains that system requirements are expanded versions of user requirements. They define the system's functions, services, and constraints in detail, and hence they should be precise. Again, as with user requirements, also the system requirements should only describe the external behaviour of the system. But this can be difficult to achieve because of the needed level of detail. The level of detail can also make the system requirements hard to understand if they are written in natural language. Hence, Sommerville (2007: 131)

suggest that more specialized notation can be used. He lists the following notations for requirements specification: stylised, structural natural language, graphical models of the requirements such as use cases, and mathematical specifications.

3.3.4. Functional and non-functional requirements

Somerville (2007: 119) states that software system requirements are typically divided into functional and non-functional requirements. He describes that the functional requirements state what the system should do i.e. what functionality it should provide. Further, these describe how the system should react on particular inputs and how it should behave in specific situations. On the other hand the non-functional requirements are constraints on the functions and services that the system offers. Somerville (2007: 122) explains that there are different types of non-functional requirements. These can be categorized into requirements related to the product, organization, and external requirements. The non-functional product requirements can be constraints regarding usability, efficiency, reliability, and portability, while the organizational requirements are requirements on the delivery and implementation, but also what standards have to be used. The external requirements are requirements regarding interoperability with other systems (interface requirements), ethical requirements, and legislative requirements that ensure the system operates within the law.

3.3.5. Domain requirements

Somerville (2007: 125–126) states that the domain requirements are derived from the application domain, and they are important because they often reflect fundamental features or attributes of it. If these requirements are not satisfied it can be impossible to produce a working system. Further, the domain requirements can be new functional requirements or constrain existing functional requirements. They can also state how specific computations should be carried out.

3.4. Requirements engineering process

Sommerville (2007: 143) states that the purpose of the requirements engineering process is to maintain a system requirements document. Further, he identifies the four main sub-processes of it: the feasibility study, requirements elicitation and analysis, requirements specification, and requirements validation, see figure 4. These sub-processes are not performed in a linear sequence. Wiegers (2003: 59) suggest that in practice, these activities are interleaved, incremental, and iterative. He continues that because of the diversity of software development projects, there is no standard approach to requirements development.

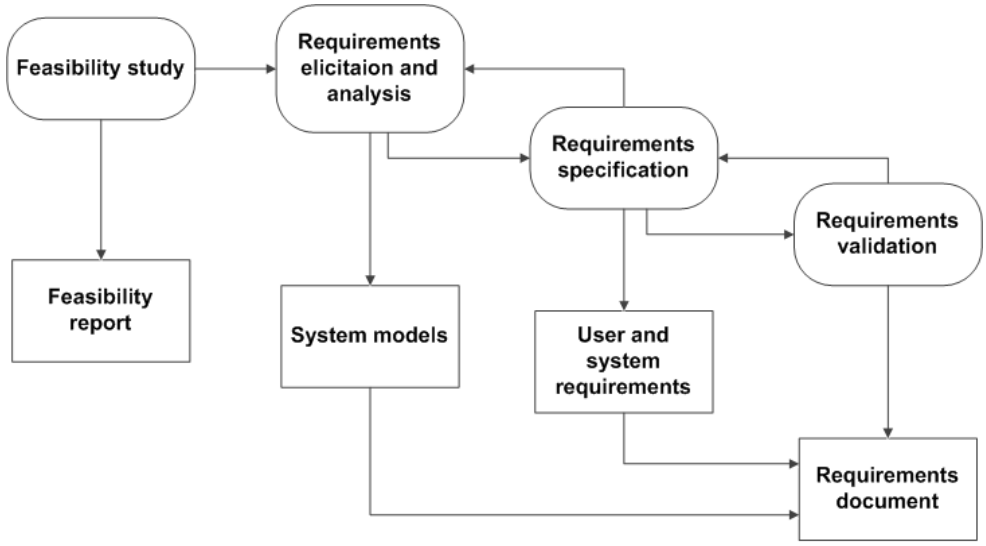

Figure 4. The requirements engineering process. (Sommerville 2007: 143)

In addition, Sommerville (2007: 143) mentions the requirements management process, which is important, because the requirements change in nearly all systems being developed. But, there also exists some confusion about the terminology. Sommerville (2007: 143) calls the entire domain requirements engineering, while Leffingwell and Widrig (2000: 16) refer to it as requirements management. Wiegers (2003: 12–13) again has found it useful to split up the domain into requirements development and requirements management, see figure 5.

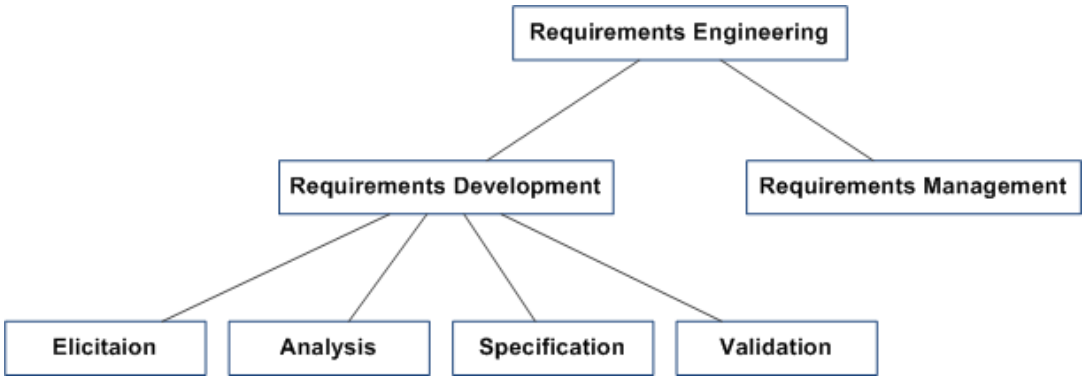

Figure 5. The requirements engineering domain (Wiegers 2003: 13).

Sommerville (2007: 143) has included the feasibility study as a sub-process of the requirements engineering process, whereas Wiegers (2003: 13) has not. Sommerville (2007: 143) summarize the sub-processes as; the feasibility study, which is a brief report weather the proposed system is useful for the business and hence if it is worth to continue with the requirements engineering and system development process. The requirements elicitation and analysis cover the requirements discovery, which is followed by the requirements specification process. The requirements specification meaning that the requirements are described and documented according to some standard. Finally, the requirements validation process covers the review of the requirements, whether they actually define the system that the customer wants. Figure 5 shows also the various documents that are produced as results of each sub-process.

The next sections will discuss the different sub-processes of the requirements engineering process in more detail.

3.4.1. Feasibility study

A feasibility study or also sometimes named as a requirements study is a short and focused study and its result should be a report that suggests whether a requirements engineering and system development process should be initiated or not (Sommerville 2007: 144-145). As inputs to the feasibility study he list; the preliminary business requirements, outline description of the system, and a statement on how the system will

support the business processes. Furthermore, he argues that it is critical that the system should contribute to business objectives, because otherwise there is no value in developing the system. Sommerville and Sawyer (1997: 66-67) also emphasises the importance of these and they further state that the business objectives are the fundamental reason for developing the system. Therefore, it is important that the business objectives are clearly stated. Additionally, Sommerville and Sawyer (1997: 66-67) list other benefits of the feasibility study; it is a low-cost way of avoiding problems that could appear later in the system development process and also, it will most likely reveal initial information sources for the requirements elicitation process.

According to Sommerville and Sawyer (1997: 67) the feasibility study involves three phases; decide what information is needed, collect the information from the key information sources, and write a feasibility report. First, critical information related to the developed system must be identified. Then, a set of questions must be developed to find that information. Sommerville (2007: 145) list some examples of questions:

1. "How would the organisation cope if this system were not implemented?"
2. "What are the problems with current processes and how would a new system alleviate these problems?"
3. "What direct contribution will the system make to the business objectives and requirements?"
4. "Can information be transferred to and from other organisational systems?"
5. "Does the system require technology that has not previously been used in the organisation?"
6. "What must be supported by the system and what need not be supported?"

Sommerville and Sawyer (1997: 67) continue that the questions should be asked only a small number of key people in the organization such as managers of departments where the system will be installed, system engineers and technical experts who can answer questions about the available technology. Sommerville (2007: 145) also adds the end-users of the system as an information source. Finally, the feasibility report should be written. It should include recommendations on whether the system development should continue, but it can also propose changes to scope, budget and schedule, and include additional high-level requirements of the system (Sommerville 2007: 146).

A feasibility study could also give misleading results, which can be caused by problems such as: missing information, user uncertainty, and premature commitment (Sommerville and Sawyer 1997: 68). The user uncertainty is caused by end-users of the system that are unsure about how the system will affect their work and this should be addressed by being explicit about the system and emphasize the positive benefits of it. Secondly, premature commitment could be caused by involved people not being totally objective because of their commitment to the idea. Hence, they might find good reasons for developing the system but miss possible arising problems. This could be avoided by using an outsider to complete the feasibility study, but then again the costs and schedule would increase because of the time he must spend on learning and understanding the organization.

Finally, Haikala and Märijärvi (2004: 37) state that the feasibility study is the most important software lifecycle phase in the sense of requirements specification. If the initial user requirements are incorrect then the final system cannot be good. They also stress the importance of finding the real user needs and a deep understanding of the problem to be solved.

3.4.2. Requirements elicitation and analysis

The requirements elicitation is defined by Wiegers (2003: 483, 485) as the process of identifying software or system requirements from different sources by using various elicitation techniques like interviews, workshops, and document analysis. Additionally, he defines the requirements analysis as at least the process of classifying requirements information, evaluating requirements for desired characteristics, representing requirements, producing detailed requirements from high-level requirements, and negotiating priorities. These all activities can involve many different stakeholders and organizations, which is considered to make the process of eliciting requirements difficult.

According to Sommerville (2007: 146) the elicitation and understanding of stakeholder requirements is difficult because of several reasons. First, the stakeholders do not know what they want from the new system and also they often have difficulties in explaining what they expect it to do. The requirements engineer has to understand the requirements stated by stakeholder in their own terms. Further, different stakeholders may express the

same requirements in different ways, which make the requirements engineers to have to discover commonalities and conflicts. He also mentions that the environment in which the analysis takes place is dynamic, and hence the importance of requirements may change, but also new requirements can emerge.

Sommerville (2007: 146–148) also presents a general process model for the requirements elicitation and analysis. He presents it as a spiral model, figure 6, and therefore suggests that the activities are interleaved as the process advances starting from the centre of the spiral.

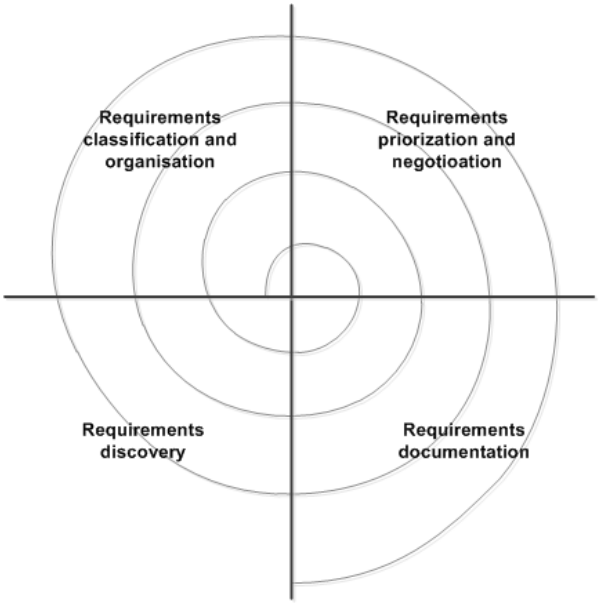

Figure 6. The requirements elicitation and analysis process (Sommerville 2007: 147).

Sommerville (2007: 146–148) continues that the requirements elicitation and analysis is an iterative process, where each activity gives continual feedback to other activities. For each round in this spiral model the analyst's understanding of the requirements will improve. The process cycle starts with the requirements discovery, which is the activity of interacting with stakeholders to collect their requirements. The requirements classification and organization activity takes the unstructured requirements and groups them into related groups of requirements. Also, the identification of overlapping requirements is an important part of this activity. Next, the requirements prioritisation and negotiation is the activity to prioritise requirements, but also about finding and

solving conflicting requirements. The conflicts should then be negotiated with the stakeholders. Lastly, in the requirements documentation stage the requirements are documented. The document could be a requirements document or a table of requirements that is completed as the process proceeds.

3.4.3. Requirements specification

Wiegers (2003: 165–167) states that the Software Requirements Specification (SRS) document is an agreement between the customer and the developer about the product to build. Hence, it must contain the functions and capabilities that a software system must provide, and also the constraints that it must respect. The SRS is an important document, which purpose is to be the basis for all subsequent project planning, design, coding, testing, and user documentation. Therefore, the system's behaviour should be described as completely as possible in the SRS, but without neither specifying design of the new software nor project requirements.

Wiegers (2003: 165–167) mentions that software requirements can be represented in several different ways. They can be written in well-structured documents in natural language, illustrated with graphical models, or written in formal specifications using mathematically precise formal logic languages, which also provide the most rigor and precision. However, the software requirements should be written in an understandable fashion so that the project key stakeholder can review the requirements and be sure on what they are agreeing to. Wiegers (2003: 167–168) gives the following suggestions on requirements readability:

- Be consistent labelling requirements, sections, and subsections.
- Leave text ragged on the right.
- Use white space liberally.
- Consistent and realistic use of visual emphasis (bold, underline, italics, and different fonts).
- Help users to find the information that they need by using a table of content and perhaps an index.
- Number and give captions to all figure and tables, and refer to them by number.
- Use the word processors cross-reference facility to refer to other locations within a document.

- Use hyperlinks to jump between related sections in the SRS or in other documents.
- Use a suitable template to organize the information.

The IEEE Recommended Practice for Software requirements Specifications (IEEE Std 830-1998 1998) is a SRS template that is according to Wiegers (2003: 171) suitable for many kinds of projects. Anyway, he suggests that it should be modified to fit the needs and nature of a project. Figure 7 lists the by IEEE Std 830-1998 (1998) proposed table of contents for a SRS document.

1. Introduction
   1.1. Purpose
   1.2. Scope
   1.3. Definition, acronyms, and abbreviations
   1.4. References
   1.5. Overview
2. Overall description
   2.1. Product perspective
   2.2. Product functions
   2.3. User characteristics
   2.4. Constraints
   2.5. Assumptions and dependencies
3. Specific requirements
   3.1. External interface requirements
   3.2. Functional requirements
   3.3. Performance requirements
   3.4. Design constraints
   3.5. Software system attributes
   3.6. Other requirements

Figure 7. IEEE Std 830-1998 (1998) proposed table of contents for a SRS document.

The introduction section in the IEEE Std 830-1998 (1998: 11–12) template gives an overview of the entire SRS document. It should state the purpose of the document, its

scope, definitions, acronyms, abbreviations, and references in the SRS to other documents. Finally, it should also include a description of what the rest of the SRS contains and how it is organized.

The overall description section presents a high-level overview of the product. The IEEE Std 830-1998 (1998: 12–15) states that this section should not state the specific requirements, but instead it should provide background for the requirements specified in section 3 of the SRS. First, the subsection product perspective should describe the context in which the product will be operated. If the product is part of a larger system, also the major interfaces should be identified here. Next, the product function should list the major functions that the product will contain. Wiegers (2003: 174) suggests that a context diagram, use case diagram, or a class diagram might be useful to give a high-level summary of the features. The user characteristics should describe the general characteristics such as educational level, experience, and technical expertise of the intended users of the product. The constraints section should describe any factors that will limit the developer's options, such as the use of specific technologies, tools, programming languages or operating environment, standards, business rules, hardware limitations, user interface conventions, or safety and security constraints. At last, the assumptions and dependencies should list each of the factors that affect the requirements, i.e. factors that can cause that changes to the requirements in the SRS are necessary.

According to the IEEE Std 830-1998 (1998: 15–16) the specific requirements section should contain all the requirements at an adequate level of detail to enable the designers to design a system to satisfy those requirements. This section is often the largest and the most important part of the SRS. First, it should include the external interface requirements that describe all inputs into and outputs from the software system to ensure that the system will communicate properly with external components. Next, the functional requirements should define the fundamental actions that must take place in the software. The IEEE Std 830-1998 (1998: 21–26) gives many possible ways to organize the functional requirements; by use case, mode of operation, user class, stimulus, response, object class, or functional hierarchy. According to Wiegers (2003: 175) an approach that makes it easy for readers to understand the product's intended capabilities should be selected.

Also, non-functional requirements other than external interface requirements such as the performance requirements, design constraints, and software system attributes should be stated in the specific requirements section. The software system attributes include requirements regarding reliability, availability, security, maintainability, and portability. Finally, the other requirements section should define all the requirements that are not covered elsewhere in the SRS. Wiegers (2003: 180) mentions that e.g. internationalization and legal requirements could be stated here. But also subsections for requirements regarding operations, administration and maintenance, product installation, configuration, start-up and shutdown, recovery and fault tolerance, and logging and monitoring could be added here.

Wiegers (2003: 190) also suggests that a data dictionary should be included in the SRS or as an appendix to the SRS. A separate data dictionary makes it easy to find primitive data elements and data structures, and it also avoids redundancy and inconsistency.

3.4.4. Requirements validation

Sommerville (2007: 158–160) define the requirements validation as the process of showing that the requirements define the actual system that the customer wants. Requirements analysis and requirements validation have overlapping functions such as to find problems with the requirements. Also, extensive and costly rework in the later phases of the development process can be avoided, if the requirements have been properly validated. Furthermore, Sommerville (2007: 158–160) has listed important checks that should be made for the requirements in the requirements document.

1. *Validation checks*. Further thought and analysis could identify more or different functionality that might be needed.
2. *Consistency checks*. There should not be any conflicting requirements like contradictory constraints or descriptions of the same system function.
3. *Completeness checks*. Requirements for all intended functions and constrains should be included in the requirements document.
4. *Realism checks*. To ensure that the requirements can be implemented, also taking into account the budget and time schedule.
5. *Verifiability*. The requirements should be written so that they can be verified by writing a set of tests.

The requirements can be validated by using one or combing several different validation techniques. Sommerville (2007: 159–160) mention the techniques: requirements reviews, prototyping, and test-case generation. In a requirements review a team involving people from both the client and contractor side analyses the requirements systematically for anomalies and omissions. In prototyping the end-users can experiment with a model of the system and see if it meets their needs. As the last technique he mentions the test-case generation, which often reveals requirements problems. In this approach a requirement should be reconsidered if it is impossible or difficult to plan a test for it. Finally, Sommerville and Sawyer (1998: 190) mention that the detected problems with the requirements have to be dealt with by re-entering the earlier phases of requirements engineering.

3.4.5. Requirements management

According to Sommerville (2007: 143) the requirements management process is important, because the requirements change in nearly all systems being developed. The changes that have to be managed can relate to modifications in the system's hardware, software, or organizational environment, but also to the stakeholders understanding of the problem.

3.5. Unified Modelling Language

"The Unified Modelling Language (UML) is a general-purpose visual modelling language that is used to specify, visualize, construct, and document the artifacts of a software system" (Rumbaugh, Jacobson and Booch 2004: 3). The original purpose with UML was to unify the earlier modelling techniques and experiences to a standard approach with today's best practices. UML is independent from any implementation technology and software development process, but still they should support an object-oriented approach to software production. UML models are used for capturing the static structure and dynamic behaviour of a system. The models consist of collaborating objects that interact to perform tasks that outside users can benefit from. Each model describe some specific aspect of the system where as other models describe other aspects. Together, they describe the complete system. (Rumbaugh et al. 2004: 3)

The different concepts and constructs in UML are divided into several views, which are a subset of the UML modelling constructs. The UML provide one or two diagrams to visualize the concepts in each view. The diagrams and views are listed below:

- Class diagram – static view
- Collaboration diagram – design view
- Component diagram – design view
- Use case diagram – use case view
- State machine diagram – state machine view
- Activity diagram – activity view
- Sequence diagram – interaction view
- Communication diagram – interaction view
- Deployment diagram – deployment view

In addition, UML provides a model management that can be used to organize the diagrams into hierarchical units. A hierarchical unit is called a package, and it gives a complete abstraction of a system from a particular viewpoint. (Rumbaugh et al. 2004: 25–27)

3.6. Use case approach to requirements gathering

Wiegers (2003: 133) explains that the objective of the use case approach is to describe all the tasks that the users need to accomplish with the system, instead of asking the users what they want the system to do. In theory, the resulting set of use cases will describe all the desired system functionality, but in practice it is difficult reach complete closure. He continues that with the use case approach it possible to get closer than with any other elicitation technique.

Also, Cockburn (2001: 13–14) states that accurately written, the use cases define what the system must do, and it should not be necessary to convert them into some other form of behavioural requirements. However, they are not all the requirements, because they do not describe e.g. the external interfaces and data formats, but they are all the behavioural requirements.

3.7. Use cases

"A use case is a coherent unit of externally visible functionality provided by a classifier (called the subject) and expressed by a sequences of messages exchanged by the subject and one or more actors of the system unit" (Rumbaugh et al. 2004: 78). Cockburn (2000: 1) states that a use case captures a contract between the stakeholders of a system about its behaviour. Further, Kulak and Guiney (2000: 46) point out that the use cases are not very detailed, and hence another tool is needed to describe the detailed interactions: scenarios.

3.7.1. Scenarios

Kulak and Guiney (2000: 46–48) have identified several definitions for the term, but they summarize scenarios as instances of use cases that effectively test one path through a use case. According to Cockburn (2000: 27–29) a use case is a collection of all the scenarios presenting the ways it can end with success or failure. He continues that each scenario is a consecutive description for one set of circumstances with one outcome. Moreover, scenarios can also contain sub use cases as its steps. In this case the only thing that matters to the scenario is whether the sub use case ended with success or failure. Kulak and Guiney (2000: 48) also state that the scenarios are a useful tool for testing the validity of use cases in an early phase of the lifecycle.

Schneider and Winters (1998: 30–31) call the most common sequence of steps through a use case as the primary scenario, while Cockburn (2000: 87) calls it the main success scenario. Every use case must have a main success scenario, which describes the typical and easy-to-understand flow of events as if everything goes right. But, the world is not perfect, and hence alternative flows of events are needed to handle e.g. branching and errors. Again, there is variation in the naming and writing convention of these. Schneider and Winters (1998: 30–31) call these alternative paths as secondary scenarios, while Cockburn (2000: 99–106) describes these as extensions of a use case. He continues that the extensions are where the most interesting system requirements reside. The extension conditions are the conditions under which the main success scenario takes a different path that can either end in success or failure of the use case.

The handling of these extension conditions is taken care by an extension handling that is like a miniature use case.

3.7.2. The actor

"An actor is an idealization of a role played by an external person, process, or thing interacting with a system, subsystem, or class" (Rumbaugh et al. 2004: 77). The actor that can be a human, some other computer system, or process, is not part of the system itself. Each actor has a specified role that it acts towards the system. One physical user may play the role of several actors towards the system. On the other hand, several different users might represent the same actor, because they play the same role with regards to the system. An actor can interact with one or several use cases by exchanging messages. In UML symbols, an actor is drawn like a stick person. (Rumbaugh et al. 2004: 77)

Cockburn (2001: 58) suggest that the actors should be listed in an actor profile table that contains the name of the actor and the actor's background and skill. By having this list the developers will get a better understanding of how the software will suit the needs of the end users.

3.7.3. The use case diagram

Maciaszek (2001: 48, 51) state that a use case diagram is a visual representation of actors and use cases, but it is also a fully documented model of the system's intended behaviour. He also argues that the use case diagram is the most important visualization technique for a behavioural model of a system. Figure 8 shows a telephone catalogue sales application depicted as a use case diagram.

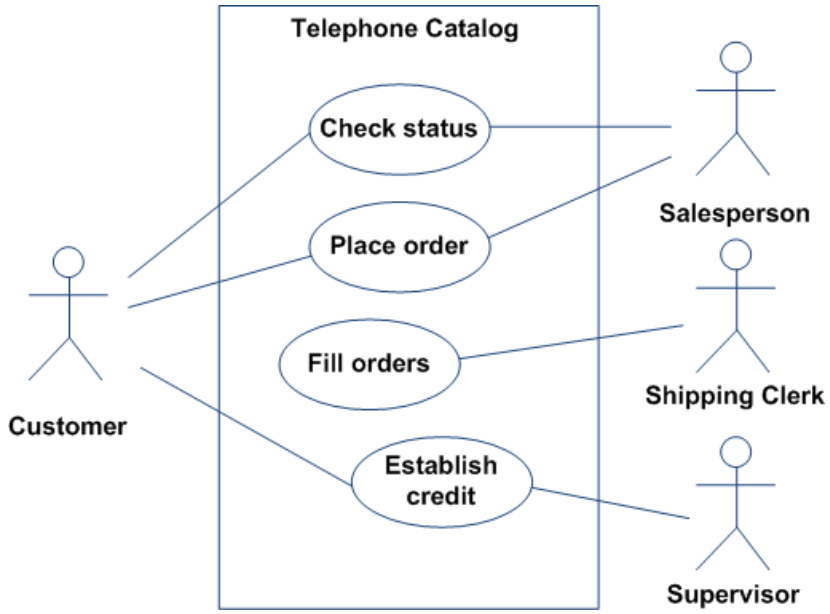

Figure 8. Use case diagram (Rumbaugh et al. 2004: 78).

In UML a use case is drawn as an oval with a short title inside the oval or below it. The use case diagram assigns actors to use cases. This is drawn by using a solid line that connects the actors with the use cases that it can communicate with. The subject boundary, system boundary, is drawn as a rectangle separating the use cases and the actors. The subject name is also stated inside the rectangle. (Rumbaugh et al. 2004: 78–80).

UML offers several relationships that a use cases can participate in. First, the association relationship is the communication path between an actor and a use case and it drawn as a solid line, as can be seen in figure 8. Next, the behaviour of other use cases can be incorporated in a use case by using the include relationship. By using the include relationship large use cases can be split into simpler use cases that describe fragments of the large use case's behaviour. Next, the extend relationship can be used to add behaviour into a base use case by extending it to another use case. A base use case can be extended to several other use cases to describe behaviour that the base use case does not know about. The included use case is necessary for the base use case to complete, whereas the extended is not. A dashed arrow including the "<<include>>" and "<<extend>>" text is used to draw these relationships. In the case of including a use case the arrow is pointing on the included use case, but when extending a use case it

point on the base use case, see figure 9. At last, the aspect of a use case can be specialized using generalization. A specialization of the parent use case is called a child use case. Use case generalization is drawn with an arrow with a large triangular arrowhead that is pointing on the parent use case. The different relationships are depicted in figure 9, where the "Place order" use case is split into simpler use cases. (Maciaszek 2001: 135–136, Rumbaugh et al. 2004: 78–80)

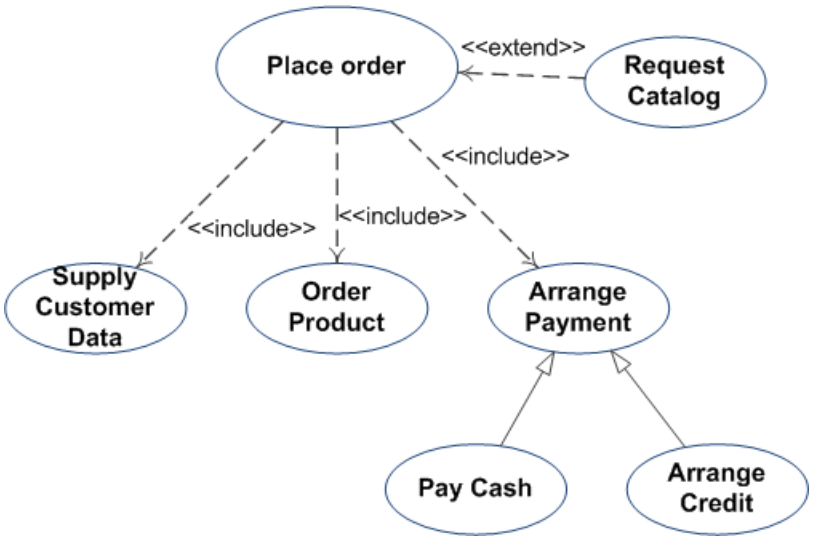

Figure 9. Use case relationships (Rumbaugh et al. 2004: 80).

3.7.4. Documenting use cases

Maciaszek (2001: 52) state that the flow of events between a use case and an actor has to be textually described in a use case document. He continues that this document will evolve with the development progress. First, only brief descriptions are written and later on also other parts of the document will be gradually and iteratively completed. At the requirements specification phase the document will be complete, and prototypes of the graphical user interface can be added. In the end, the user documentation for the new system can be produced from the use case document.

3.7.5. Writing use cases

According to Cockburn (2000: 2) a well written use case is easy to read, and learning to read them should not take more than a few minutes. On the other hand, to write use cases is harder. He claims that the writer has to master the following concepts:

- Scope: What is the system under discussion?
- Primary actor: Who has the goal?
- Level: On a how high- or low-level is that goal?

Then the writer must apply these concepts to every sentence in the use case, but also to the use case as a whole.

Cockburn (2000: 35–38) defines the word scope as the extent of what is designed. He further introduces the concept of functional scope and design scope. The functional scope refers to the services that the system offers, while the design scope is the extent of the system i.e. the boundary of the systems, hardware, or software that is designed. He continues that a very simple way to manage scope discussion is to keep an in/out list. It is a very simple list with three columns named: topic, in, and out. The in/out list should be used for both functional scope and design scope. Other tools that he proposes for managing the functional scope are the actor-goal list and the use case briefs.

According to Cockburn (2000: 36–37) the purpose of the actor-goal list is to show the system's functional content by listing all user goals. While the in/out list showed both in and out of scope topics, the actor-goal list present only services that in fact will be implemented by the new system. This list should be kept up-to-date to always reflect the status of the system's functional boundary. Table 1 presents an example of an actor-goal list.

Table 1. A shortened sample Actor-Goal List (Cockburn 2000: 37).

Actor	Task-level Goal	Priority
Any	Check on request	1
Authorizer	Change authorizations	2
Buyer	Change vendor contacts	3
Requestor	Initiate a request	1
	Change a request	1
	Cancel a request	4
	…	

As a second tool to define the functional scope Cockburn (2000: 37–38) presents the use case briefs. He states that a use case brief is a two to six sentences long description of use case behaviour. The use case brief should only state the most significant activity and failures of a use case, and hence its purpose is to recall people of its content. He also mentions that in project teams with extremely good internal communication and discussion with users, the use case briefs can be enough as requirements. The rest of the requirements could then be in continual discussions, prototypes, and frequently delivered increments. The use case briefs can be documented as a separate table, see table 2, but also as an extension to the actor-goal list.

Table 2. A shortened sample of use case briefs (Cockburn 2000: 38).

Actor	Goal	Brief
Production staff	Modify the administrative area lattice	Production staff adds administrative area metadata (administrative hierarchy, currency, language code, street type, etc.) to the reference database. Contact information for source data is cataloged. This is a special case of updating reference data.
	…	

To avoid misunderstandings concerning the design scope Cockburn (2000: 38–40) recommend that every use case should be labelled with its design scope. The most significant design scopes should have specific names. Further, he states that the design scope can be of any size e.g. enterprise, system, or subsystem. Hence, Cockburn (2000: 51) suggests that the design scope drawing should also be a work product that binds the

system's scope. The purpose of the design scope drawing is to showing the people, organizations, and systems that will interact with the system to be developed.

According to Cockburn (2000: 2) the second concept that a writer has to master when writing use cases is the primary actor. He defines the primary actor as the stakeholder who or which needs to achieve a goal with respect to the system. Most often the primary actor triggers the use case. Cockburn (2000: 54–57) also state that there are some exceptions to this. One is e.g. when a clerk or phone operator initiates the use case on behalf of someone else. Another is when the use case is triggered by time. Further, he states that the primary actors are important at two occasions of the system development; at the beginning of the requirement gathering and just before the system is delivered. At the beginning they are important because by brainstorming the primary actors, this sets up a work structure that will help to capture all the goals. Further, a list of primary actors focuses on the people, who are going to be the users of the system. Just before delivery of the system a list of all the people and which use cases they will run is needed to: package the system into units for various users, set security levels, and to create training for various user groups.

As the third concept that a use case writer has to manage Cockburn (2000: 2) mentions the goal levels. Cockburn (2000: 61–62) also states that giving names to goal levels help. He defines the levels as summary goals, user goals, and sub-functions. In figure 10 is an extract of his example of use cases at these levels and the hierarchy of goals that the use case set reveals.

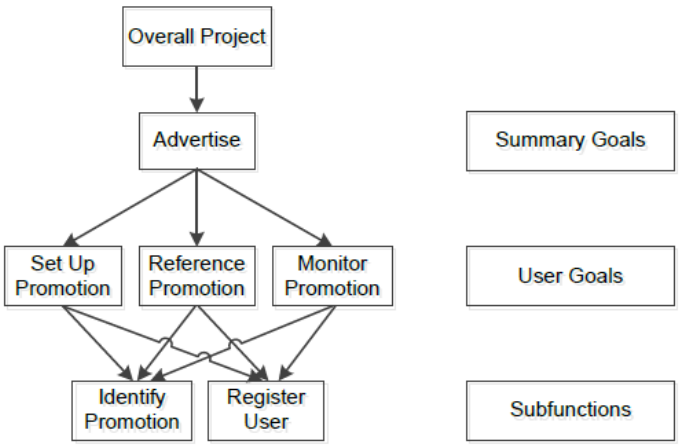

Figure 10. Use case levels (Cockburn 2000: 62).

Cockburn (2000: 62–67) states that the user goals are of the greatest interest, and hence the most energy should be put on finding these. Further, the user goals are goals that the primary actors are trying to achieve. He continues that these should pass the one sitting test, meaning that a goal should be achieved without taking any brakes and within 2-20 minutes. On the other hand, Summary-level goals that provide the context where the user goals will operate, can execute over hours, days, weeks, months, or years. There should only be a few of these use cases. The summary-level goals involve many user goals, and hence their purpose is also to show the life-cycle sequencing of related goals. Further, the summary goals can also be considered as a table of contents for lower level goals. Finally, the lowest level of goals, the sub-functions are according to Cockburn (2000: 66) goals that are needed to carry out user goals. These should only be used to improve readability or because many other goals use them.

Writing use cases requires skill and practice and hence inexperienced developers and analyst's writing use cases for the first time often run into similar kinds of problems. Lilly (1999) has collected experiences from real projects using use cases, and from these listed the "Top Ten" problems and pitfalls. A summary of these problems and pitfalls can be found in table 3.

Table 3. Top-ten problems and pitfalls when writing use cases (Lilly 1999).

Problem #1:	**Undefined or inconsistent system boundary.**
Symptom:	The use cases are described on different scopes like business, system, or subsystem scope.
Cure:	It should explicitly be specified at which scope the use cases are defined.
Problem #2:	**The use cases state the system's point of view instead of the actors'.**
Symptom:	The names of the use cases describe system functions instead of goals that the actor wants to achieve e.g. "Process Ticket Order" and "Display Schedule".
Cure:	The use cases should be named from the perspective of the Actor's goals e.g. "Order Tickets" and "View Schedule".
Symptom:	Internal functionality is described in the use case specification steps instead of interactions across the system boundary.
Cure:	Concentrate on what the system needs to do so that the actor's goals will be satisfied, not how it will accomplish it.
Symptom:	The use case model appears to be more like a data/process flow diagram.
Cure:	Be careful with the <<include>> and <<extend>> use cases and make sure that they describe interactions between the actor (base use case) and the system.

continues...

…continues

Problem #3:	**Inconsistency in actor names.**
Symptom:	The same role is described by several different actor names.
Cure:	The used actors should be defined and agreed upon early in the project. Also a glossary with the actor's name, its meaning, and any alias that it is known by should be established.
Problem #4:	**There is too many use cases.**
Symptom:	There are a large number of use cases in the use case model.
Cure:	The granularity of the use cases should be suitable. Combine use cases with trivial and occasional behaviour that might accidentally been chopped into fragments. Remove use cases that describe "internal" system processing in relation to the system boundary being used. If the system is very large, then the use case model should be partitioned into packages. Each package should contain a limited number of actors and describe a uniform set of use cases.
Problem #5:	**The use case model reminds a spider's web.**
Symptom:	The use case model contains too many actor-to-use case relationships, actors that interact with every use case, or use cases that interact with every actor.
Cure:	The actors might be too roughly defined and hence it should be considered if more specific roles could be defined. In turn, in some cases a more general class of actors could simplify a model. A use case model could be simplified and redrawn by using actor generalization, which purpose is to identify similarities of actor roles.
Problem #6:	**Too long use case specifications.**
Symptom:	The specification for a use case goes on for pages.
Cure:	Either the granularity of the use case is too rough or the granularity of the steps might be too fine. If the steps in the use case are too detailed, then they should be rewritten by concentrating on the essential interaction.
Problem #7:	**Confusing use case specifications.**
Symptom:	The context of the use case is missing.
Cure:	Relevant set of circumstances with respect to the "big picture" should be added in a context field in the use case specification.
Symptom:	The steps in the use case specification looks like a computer program.
Cure:	The focus should be on describing the essential interaction between the system and an actor, which results in achieving the actor's goal. Alternate flow should be used for conditional behaviour. Complex algorithms should be described using other more effective techniques. Do not specify implementation in the steps.
Problem #8:	**Functional entitlement incorrect described in use case.**
Symptom:	The relationship between the use cases and the actors do not describe what each actor can do with the system. E.g. one actor can use the normal flow but not the alternate flow in a use case. The problem could appear if the use case writer has tried to be "object-oriented" and therefore included all possible actions like create, read, update, and delete in a use case. Secondly, this problem could also occur if the developer has tried to match use cases to the interface screen.
Cure:	Verify that all actors associated with a use case are allowed to perform it entirely. Otherwise, the use case should be spit.

continues...

…continues

Problem #9:	**Customers do not understand the use cases.**
Symptom:	A customer that needs to review or approve a use case-based requirements document does not know anything about use cases.
Cure:	The customers should be taught just enough about use cases. A short explanation should be added in the use case document, which would include a key to read the model and specifications together with an example. Also a short training session could useful before the document should be reviewed. <<include>> and <<extend>> relationships should be advisedly used.
Symptom:	The use cases do not tell a story.
Cure:	Add a context field in the use case specification and an overview section for a set of use cases e.g. a package, which would tell the story. Also, consider to use other kinds of models to complement the use case.
Symptom:	The customer thinks of the problem in another way than how the use cases are organized.
Cure:	Listen to the customer and figure out how the use cases could be better organized e.g. by using packages to describe major roles/actors or events in the customer's business. The use cases should also be ordered "chronologically" to describe a story of system use over time.
Symptom:	Use cases are written with computer slang.
Cure:	This terminology is not part of the customer's vocabulary and therefore should be avoided.
Problem #10:	**Use cases are never completed.**
Symptom:	User interface changes require use case changes.
Cure:	The use cases and user interface should be loosely coupled. The use case requirements must be satisfied by the user interface design, and not the other way.
Symptom:	Design changes require use case changes.
Cure:	The use cases should not contain design. They should record what the system must do and hence they should not be specified at a too low level.
Symptom:	Many possible alternate cases.
Cure:	Cover 80% of the cases, because at a point further analysis and specification does not add quality.

3.8. Use cases as tools of requirements specification

Kulak and Guiney (2000: 59–60) describe some of the roles that the use cases play as tools of requirements specification. First, they state that the use cases are effective communication vehicles, even better than any other document produced by IT people. The reason is that the use cases describe the most basic interaction between an actor and the application by leaving out distracting computer-specific jargon and user interface details. He continues that use cases can be used both for functional and non-functional requirements effectively. The functional requirements can be described by interactions

between an actor and the application, whereas the non-functional requirements can be stated as use case stereotypes. These definitions of stereotypes should be quite detailed, and also reviewed by users. Kulak and Guiney (2000: 59–60) also claim that use cases provide requirements traceability, because they are a building block for e.g. system design and test cases. Hence, this helps in assuring the stakeholders that all the requirements are being addressed. Lastly, use cases also constrain premature design by obviously revealing design creep.

3.9. An approach to translate user needs into user requirements

Kujala, Kauppinen and Rekola (2001: 46) state that the challenge of how to bridge the gap between the technically skilled software developers and the customers is a fundamental problem in the requirements elicitation process. They have studied this in three case studies, where their purpose was to develop new user requirements elicitation methods that would help the designers to collect user and customer needs by direct contact with the users. The tasks that they completed in the case companies where; identifying stakeholders, gathering user needs, describing user needs for use cases, documenting use cases, and gathering user feedback. Kujala et al. (2001: 47) developed a method where an initial task sequence could be translated to use cases. In their paper they suggest that the move from informal descriptions of user data towards the user requirements begins with determining the user needs. An example of the task sequence is represented in figure 11.

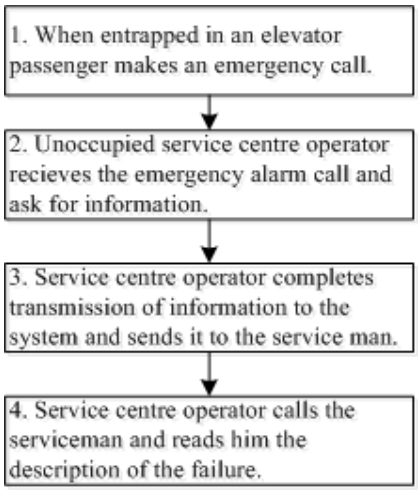

Figure 11. Task sequence diagram (Kujala et al. 2001: 47).

Next, Kujala et al. (2001: 47) explain that the task sequence steps should be linked to users' problems and possibilities. This would serve as a basis for writing the use cases. This method was found useful by several of the participants in the case studies. Table 4 extends the task sequence in figure 11 with problems and possibilities stated by the customer.

Table 4. User need table (Kujala et al. 2001: 48).

Task sequence:	Problem
Step 1: When entrapped in an elevator passenger makes an emergency call.	• Passengers want to get out of the elevator as soon as possible • All kinds of passengers must be able to make an alarm call (blind, foreigners etc.) • Sometimes passengers may make false alarms unintentionally. • Passengers may be in panic. • Passengers need instant confirmation that they have created a connection to the service centre operator and that they are going to get help.
Step 2: Unoccupied service centre operator receives the emergency alarm call and ask for information.	• Different versions and types of remote monitoring systems. • Passenger is the only information source. • Service centre operator does not notice the emergency alarm call.
Step 3: Service centre operator completes transmission of information to the system and sends it to the service man.	• Laborious phase for the service centre operator. • Simultaneous calls must be differentiated. • Serviceman cannot see all information. • Inadequate information from a site system. • Possibility: Instructions as to how to operate the system. • Possibility: Possibility to open phone line from Call Centre to the elevator.
Step 4: Service centre operator calls the serviceman and reads him the description of the failure.	• Extra work for the service operator.

The last step was to write the use cases based on user need tables and a list-based requirements document. Kujala et al. (2001: 49) decided to describe the use cases according to how Rumbaugh (1994) propose to describe them, but with some exceptions. The description of the use case they wrote with numbered steps, and they also connected the exceptions with numbers to the step that they related to. Further, they described the goal of the users in the precondition part. A simplified use case that is

written based on the user need table and a list-based requirements document is presented in table 5.

Table 5. A use case written based on the user need table and a list-based requirements document (Kujala et al. 2001: 48).

USE CASE:	Making An Emergency Alarm Call
Summary:	An entrapped passenger pushes the emergency alarm button in order to get help. A service centre operator receives the alarm call and informs the passenger that a serviceman will come and let the passenger out of the elevator.
Actors:	Passenger and service centre operator
Preconditions:	An elevator has stopped between floors and there is a passenger in the elevator. The goal of the passenger is to get out of the elevator safely and as quickly as possible.
Basic sequence:	Step 1: The passenger presses the emergency alarm button. Step 2: The service centre operator gets a visible notification about the emergency alarm call on the screen with an optional audio signal. Step 3: The service centre operator accepts the emergency alarm call. Step 4: The system opens a voice connection between the service centre operator and the passenger. Step 5: The system indicates to both the passenger and the service centre operator that the voice connection is open. Step 6: The system guides the service centre operator as to what information to ask of the passenger. Step 7: The service centre operator informs the system that the emergency alarm call is correct.
Exceptions:	Step 1: If an entrapped passenger does not push the alarm button long enough (less than 3 seconds), the system alerts the passenger with a voice announcement. Step 7: If the passenger has pressed the emergency alarm button by accident, the service centre operator informs the system that the emergency alarm is false. The system resets the emergency alarm call.
Post conditions:	The entrapped passenger knows that the service centre operator will contact a serviceman who will help the passenger out of the elevator safely as soon as possible.

The feedback that Kujala et al. (2001: 49) got was encouraging. The designers in the case companies found the use cases useful, because they gave them a high-level view of the product requirements and also helped them to identify missing undefined details. In

addition, through the use cases it was also easier to communicate functionality to the user. The designer also added that the use cases would serve as good checklists to guide the definition work but also as a base when writing instructions and test cases.

3.10. Challenges and issues with requirements gathering

Kulak and Guiney (2000: 2–3) state that many projects fail because of poor requirements. The purpose of the requirements is to lead the development towards the desired system, and then even a small mistake in an early phase could lead to a major problem during deployment. To correct an error at this phase would be both time-consuming and expensive. The people whose skills are in programming have difficulties to get the requirements right. This is because the requirements are so abstract and different from computer programs. Kulak and Guiney (2000: 3) state that the typical problems with requirements gathering are; it takes too long and the wrong things are documented, assumptions are made about things that have not yet happened, and the requirements are completed just in time just to make them over again, because of changes in the business.

Challenges in the requirements gathering process that Kulak and Guiney (2000: 11–14) have observed are; to find out and document the user needs, avoid premature design assumptions, conflicting and redundant requirements, reduce volume, and ensuring the traceability. The users often do not know what they want from the new computer system. Also, they have other things on their mind and other daily responsibilities. Here it is important to establish a strong relationship to the users, but also to have a good support from the management. The management can state the importance of the system and in this way encourage the users to participate in the requirements gathering. The challenge, when documenting requirements is to know whether all of them have been documented, but also to see what is missing. Every requirements specification tends to include premature design assumptions for several reasons. One reason could be that the people gathering the requirements do not trust the system designer, and they do not want them to mess up the design and make wrong decisions later. Another reason could

be when the developers are off-site separated from the requirements gatherers and user. Conflicting requirements can be requirements in a list that say the opposite things. To manage this problem a built-in mechanism is needed, but also something with more structure than a list could be used. Reviews are also useful when conflicts are identified. Redundant requirements can be confusing when they state almost the same thing. They also increase the number of requirements, which again decrease the likeliness of a successful development effort. Hence, it is important to reduce the number of requirements by removing conflicts, redundancy, and design assumptions. In addition, a reasonable abstraction level should be used. Also, all commonalities among the requirements should be found, and finally, the functional and non-functional requirements should also be separated. Lastly, Kulak and Guiney (2000: 14–15) stress the importance of the requirements traceability, where the requirements should be traceable throughout the lifecycle of development. They continue that unfortunately this is often not the case and hence a solid audit trail for all the activities is missing. By maintaining an end-to-end requirements audit trail, the addition of system functionality that is not required by the user and also not documented could be avoided.

Kulak and Guiney (2000: 14–18) have also looked into issues with some of the approaches that can be used to gather requirements. First, interviews are obviously needed when making a requirements specification. When user interviews are done with persons at various levels of management, then conflicting views of processes or business rules begins to appear. The reason for conflicting views is the required level of detail that is needed to build a computer system, which is greater than what is needed to run a business. Secondly, joint requirements planning sessions, where all the stakeholders are gathered together in the same room to give their inputs, can be a very valuable and significant timesaver for the requirements team. But, the issue here lays in the document produced, which is typically a list of requirements. Kulak and Guiney (2000: 15) argue that the requirements list must be replaced by something more structured and relevant that can be used and understand by both users and designer. Hence, they suggest that it should be replaced by use cases, use case diagrams, and business rules. Lastly, they consider the issues with prototypes. The concern with

prototypes is that users could pay too much attention on the details of the screen or user interface, instead of understanding the real purpose of the prototype. Further, executives could have difficulties in understanding how it can take another year to build a system that look as it would be ready. Also, because the prototype only includes the front end of the system, the business rules are not usually represented in it at all. Hence, prototypes should come later in the development and be used for interface specification, as that is what they are best used for.

4. THE CURRENT STATE OF CBM PRODUCTION

This chapter will focus on describing the current state of CBM production as is today. First, some background information on how the CBM production management has evolved till today will be given. Then the CBM centre will put in context to the stakeholders and also their inputs and needs to the production process will be discussed. Next, the high level CBM process will be presented. Also the tools that are used today to manage the production process will be identified and discussed. Further, the problems of today's production process will be listed and analysed. Finally, the need for a new CBM production management tool is motivated.

4.1. Background

From the beginning of the CBM centre in year 2001 until today, the CBM installation base has grown to more than 400 connected installations worldwide. This number is estimated to continue increasing, which again puts pressure on developing the CBM production process and tools. It is more and more difficult to manage with only tools like Microsoft Excel to maintain large amounts of information that is depending on information from many different sources.

Today, the CBM service is always sold as a part of some long term service agreement, but also in some cases given to the customer for the warranty period. These all involve more information than what was before needed for the CBM deliveries. Hence, it is today essential to know about the agreement dates, scope i.e. to know what has been promised to the customer and hence to know what to deliver, but also to whom to send the CBM reports and when.

All CBM services will be invoiced today, opposite to earlier practice, which means that a purchase order is needed for all CBM activities. This requires the CBM personal to be aware of order numbers on which to book hours, but this also involves managing the information of how much to invoice and when to invoice. Further, to know how much to invoice, also the number of made CBM report needs to be tracked and reported. As each purchase order has a start and end date, this has to be followed-up, and when the end date is closing in the CBM personal has to ask for a new purchase order. Otherwise,

the CBM service will be ended. This way-of-working and all this increase in information requires much manual work to manage it and to keep it all up-to-date. Additionally, all the information has to be collected from various sources manually.

4.2. Stakeholders

The CBM production process involves many stakeholders that place requests for CBM services or interact in some other way with the CBM centre. Figure 12 shows the stakeholders and their interactions with the CBM centre. The figure includes also information about what kind of requests or information exchange occurs between the stakeholders and the CBM centre. For clearness, only relevant information concerning this project has been included in this figure.

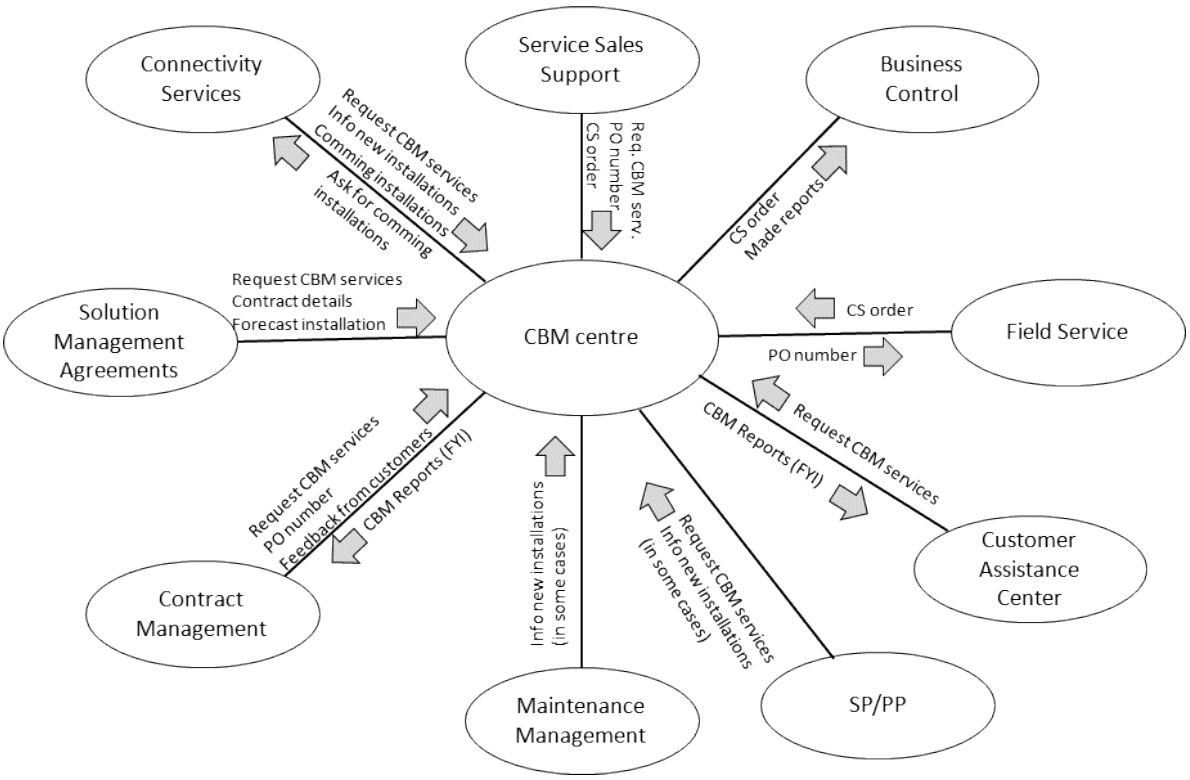

Figure 12. Stakeholders and interactions.

The interactions clearly indicate that the request for CBM services comes from many directions. Another important observation is the flow of PO numbers and CS order numbers. Today, the CBM centre coordinates the collection and distribution of these numbers and this work is very time consuming, because it all is done manually. This was one of the main reasons why this project was started.

4.3. CBM delivery process

The CBM delivery process can be divided into two main processes; delivery of the CBM system and delivery of CBM services. The delivery of the CBM system is a once occurring project where the system itself is delivered to the customer, while the delivery of CBM services is on-going CBM work until the service is ended. As shown in the figure 13 the main process can further be divided into the sub-processes offer, plan, deliver, and maintain.

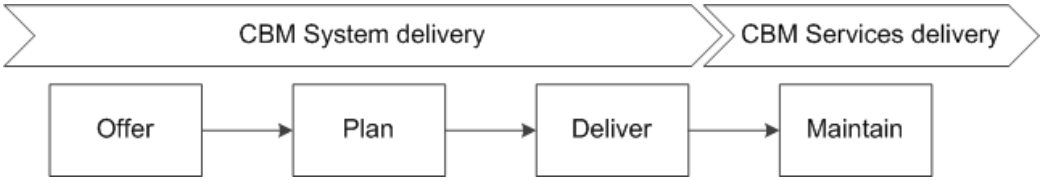

Figure 13. The CBM delivery process.

The CBM delivery process begins with the offer phase where Sales offer CBM to a customer. Typically, CBM is an important part of long term service agreements, but it can also be offered to a customer e.g. for the warranty period of the engine. As an agreement is signed also a CBM order will be created. When the CBM Connectivity Services has received a purchase order for the CBM system, the planning of the CBM system delivery can start in the plan phase. Activities in this phase are to ensure feasibility of the delivery, order hardware and software, reserve resources for commissioning, and communicate with customer about delivery details. Next, in the delivery phase the system is delivered to the site and a field service engineer commissions it. Also, a connection is established and initial data is transferred to the

CBM central database, which also has to be configured. Finally, the data is validated, after which the system is ready for the delivery of CBM services.

The maintain phase, consists of continuous CBM work. The data that is received on a daily basis from the site is first automatically analysed, where after the engine experts will review it and write monthly reports for the customer. Further, the reports have to be invoiced. Invoicing occurs once or twice a year according to what has been agreed with the customer. Because the invoicing periods vary from one installation to another, this has to be followed up, in order to know when to invoice each customer. In addition, only the actual number of sent reports will be invoiced, and therefore also this needs to be considered when invoicing the reports. For each invoicing period a separate purchase order is needed, which involves work to request it and forward it to a Service Coordinator, who can open a customer service order for it. This is needed for cost assignment. Finally, also the contract dates has to be followed up so that the customer can be notified about when the CBM service will be disconnected.

4.4. Tools used today

The tools used today to manage the delivery of CBM are basically Microsoft Excel and Microsoft Access. Also the operative analysing tool includes some information that is needed for the CBM production management. First, for the plan and delivery phases of the CBM process, the Connectivity Services team uses a standalone Access database. The main purpose of the database is to keep track of the delivery of the CBM system. This database includes some basic information about the installation, while most of the content is related to information about the CBM system to be delivered, how to connect it to the automation system, coordination of delivery e.g. dates, place, accommodation, and resources.

The CBM team, who is responsible for the maintain phase of the CBM process, uses a standalone Microsoft Excel sheet to manage the delivery of CBM services. This Excel sheet, which also serves as the installed base file, includes first information about the installation: id, name, country, area, engines, engine reference type, configuration, and output. Secondly, the CBM team has to control invoicing of CBM services i.e. CBM reports. For this, information is needed about invoicing period, time, and price, but also the PO numbers and CS order numbers. Also, contractual information is needed to keep

track on the scope of delivery and whether to continue with CBM reporting or not. If a contract is ending, also the CBM reporting should end. The needed contract information is: contract start and end dates, scope, and contract contacts. As both tools are standalone and not connected to any other system all the contents must be manually entered into the tools. Similarly, if someone needs some of the information in these tools, it has to be requested from each tool's responsible team.

4.5. Identified problems

Table 6 lists the main problems concerning the CBM production today. The problems are also prioritized so that the most critical problem appears at the top of the list.

Table 6. Listing of identified problems in CBM production.

Priority	Problem	Description
1	**Reduced customer satisfaction**	Delays in deliveries or even total loss of deliveries are resulting in reduced customer satisfaction.
2	**Shortage of resources**	Number of installations is increasing faster than the required number of resources to handle the increased work load.
3	**Low or no predictability of coming CBM installations**	No or little information is received from other stakeholder about budgeted, offered and sold agreements that has CBM included.
4	**Loss of invoicing**	Delays in invoicing because of missing PO numbers.
5	**Delays in deliveries**	Delays in deliveries because of missing or wrong information e.g. about contact persons. This is also a cause related to problem no. 1.
6	**Delays in invoicing**	Delays in invoicing because of missing PO numbers.
7	**Unplanned and difficult cost transfer of activities**	Much work is needed to maintain the information related to cost transfers up-to-date.
8	**Considerable amount of manual work required to keep contract and invoicing information up-to-date**	All information is entered manually into an excel sheet which in turn require a considerable amount of manual work. Further, all changes and updates also need to be checked and updated in this excel file, which easily might lead to inconsistent information.

continues…

…continues

9	**Contract, invoicing and customer information not up-to-date**	Information that is not up-to-date in the excel file is leading to delays in deliveries and invoicing.
10	**Extract reference lists and statistical data**	Difficult to extract reference lists and statistical data from an excel file.
11	**Find information**	Difficult to find correct information easily in excel file.
12	**Internal information (e.g. CBM connected installations)**	Futile work to search internal information that could be more widely available for requestors. Particularly, when the information is only available for a small group of people.

A total of twelve problems were identified, and a glance at the table will quickly reveal that almost all of them are in some way related one to another. Reduced customer satisfaction has been identified as the number one problem. This reduction in satisfaction is caused by the fact that deliveries are delayed or even worse, totally lost. Several reasons to this can be identified; the first one can be found in problem number five that states that the missing or wrong information can be the cause. Further, this also relates to problems stating that information is not up-to-date. In addition, the information might be outdated because of a too high work load, which is stated in problem number two, the shortage of resources. The source to the high work load could be that a considerable amount of manual work is needed to keep information up-to-date. Additionally, this will also reveal that the too high work load is a cause of the poor predictability of coming CBM requests. As it can be notice, the interconnection of most of the problems is obvious.

As all problems are tight interconnected with each other, this strengthens the need for a solution. A solution, that could improve customer satisfaction, reduces workload, and gives better predictability of coming CBM installations. Further, a solution with up-to-date information, updated with a minimum amount of manual work. Finally, this information would also have to be easy to locate and simple to share. In the next section the need for a new tool is motivated.

4.6. Why is a new tool needed?

The problems listed in the previous section would remain if a new tool is not developed. In addition, these problems will continue to grow in magnitude as the number of CBM installation continues to increase. In the worst case, if the tool would not be developed, additional resources would be needed to manage the administrative tasks like invoicing of CBM reports and completion of CBM information into the standalone applications that are used today. Further, the CBM organization would have difficulties to respond to the growth of sales. It would not be possible to deliver CBM systems and services on time to all new signed contracts, without increasing headcount. As the major part of the new CBM installations are sold as a part of some service agreement, also the amount of information to manage has increased.

First, a common tool with up-to-date information would serve the involved stakeholders better than many small standalone applications. A common tool would also improve the transparency and collaboration, because everyone could utilize the tool to check information concerning CBM installations, but also the status of on-going and coming CBM deliveries. Compared to the situation today, the tool data should have the possibility to be updated automatically from the organization's other systems. This could be realized by interfaces to or integration of the tool into existing applications that contain information needed for CBM delivery.

One of the main problems with today's process is the lack of transparency. This is a result of the CBM product portfolios and teams using own standalone applications, and hence the information is not available to other stakeholders. This results in lack of information about coming CBM installations, which is a remarkable disadvantage concerning the planning of both future resources and daily work.

The most important business objective that a new tool would serve is to enable the growth of sales, especially the sales of new agreements. Other important identified business objectives are to serve the customer better and also to enable better resource planning within the CBM related activities. As this would be a global tool for managing CBM production it would also harmonize the way of working but also enable transparency of CBM work between the CBM portfolios. By achieving these business objectives the collaboration between the stakeholders would be improved, also resulting

in an easier coordination of CBM deliveries where more than one CBM portfolio is involved.

There are some important features that the new tool should support.

- Order CBM system and services
- Manage CBM deliveries (CBM system and services)
- Enable forecasting of CBM deliveries
- Manage invoicing activities
- CBM information up-to-date and in one place
- Provide easy accessible CBM information to all stakeholders

5. THE REQUIREMENTS DISCOVERY

The requirement gathering is an activity of the first phase of the Wärtsilä Project Model, the initiate phase. However, as the operational development project's start was very much delayed, I performed the requirements discovery for this thesis already in the explore phase. The scope for the operational development project was to develop a production management tool for CBM services. This included also other CBM portfolios such as 2-stroke and propulsion. But, because of the delay, I limited the scope for this thesis only to requirements gathering for the CBM 4-stroke portfolio.

The main tasks for the requirements discovery for the Production Management Tool (PMT) were to interview and have discussions with stakeholders, investigate as-is process, documents and tools, and finally document the requirements in a Software Requirements Specification.

5.1. Scope

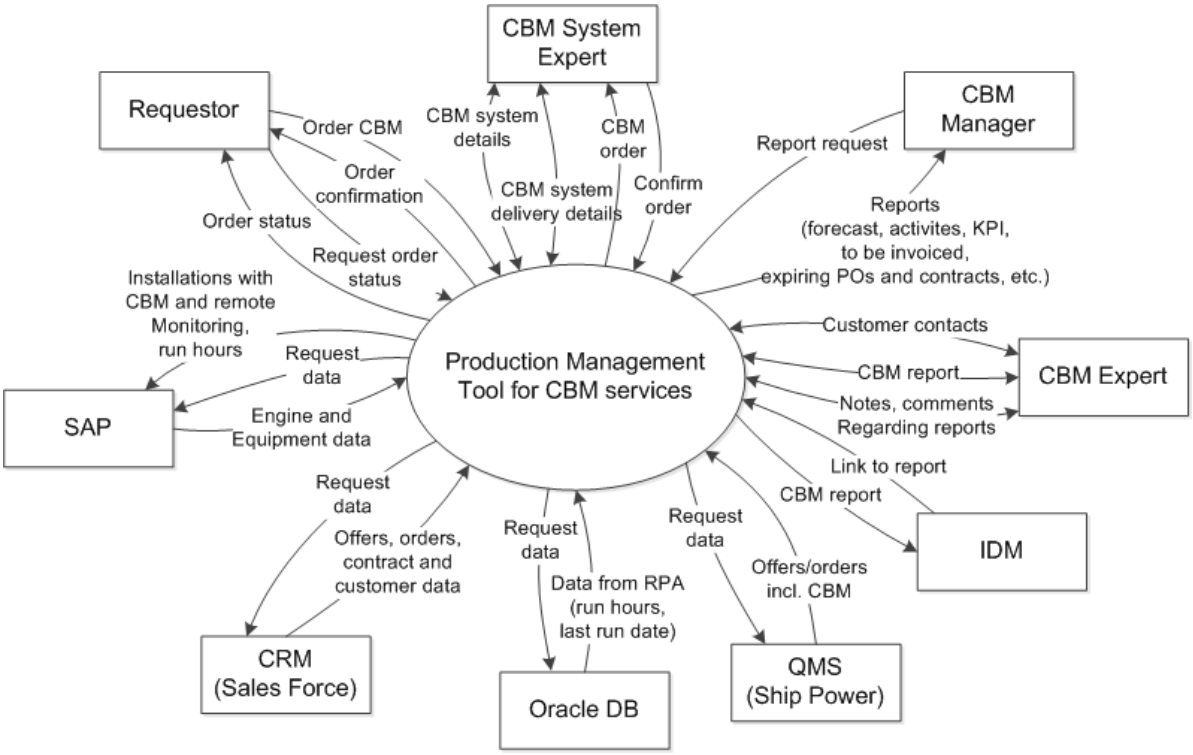

Figure 14. Context diagram of the production management software.

To define the scope of the new software I primarily used a context diagram, see figure 14. The context diagram was practical because it gave an easy way to communicate the scope to other stakeholders. In this context diagram I considered the PMT as a black-box, and hence only the interface to the outside was visible. Later, when writing the functional requirements, it was easy to verify against this diagram if they were in or out-of-scope.

When reading the context diagram, it has to be noted that some arrows are bi-directional, meaning that the information is flowing in both directions. In addition, the reader should also note that a *report* is not a *CBM report*. The information that the CBM Manager requests from the software are e.g. reports regarding forecasts of coming CBM installations, monthly activities for the monthly reporting, invoicing dates, and expiring POs and contracts. A CBM report again is the product that a CBM Expert produces with other operational applications, and then distributes with the production management tool.

In addition to the context diagram, I also used an in/out list of features to track scope changes. In the in/out list I listed the main features of the software and if this feature would be in scope. Features that were removed from the scope were not removed from the list, and in this way it was possible to track scope changes. The in/out list is presented in table 7.

Table 7. In/out list of features for the production management software.

Topic	**In/Out**
Configure and view reports (ref. lists, activities, forecast, invoicing etc.)	In
Manage contacts	In
Save and distribute CBM report	In
KPI in any form	Out
Ordering of CBM system and services	In
Definition of CBM products	Out
Collect customer feedback and satisfaction	Out
CBM system delivery planning	In
Interface to business applications SAP, CRM and QMS	In
Interface to Opera DB	In

5.2. User classes

I grouped the intended users of the production management software to a few user classes. Then I described the general characteristics of each user class in a table. Table 8 contains a list of the user classes.

Table 8. User classes of the production management software.

User class	Description
CBM Manager	The CBM manager uses the PMT to plan and report CBM activities. For these purpose he needs to get forecasts of coming CBM installations, reports of monthly CBM activities, and KPIs. He also follows-up PO and invoicing activities. The CBM manager might not necessary have a technical background.
CBM System Expert (incl. Connectivity Services)	The CBM System Expert is a technically experienced user whose expertise is in automation and information technology. He can receive orders for CBM systems and services in the PMT. He depends much on the information in the tool concerning CBM hardware, system delivery planning details, and commissioning details. The CBM System Expert also configures new CBM installations into the CBM analyzing system and do troubleshooting. Much of the information in the PMT is inserted and maintained by the CBM System Experts.
CBM Expert	The CBM Expert is technical person whose expertise is in mechanical engineering. He is typically performing analysis of engine operation. He uses the tool to store customer contacts and for distributing CBM reports. He can also add comments regarding reporting or feedback from the customer to the PMT. It is important to have the customer contact information up-to-date if someone else have to take over his work e.g. during vacation periods.
Requestor (Contract Managers, Warranty Managers, SP, PP)	The requestor can be an Area Contract Manager or a Sales person. He can order CBM systems and services through the PMT, but also follow-up delivery status. The background and expertise of the requestor can be very much varying.
Administrator	IM person or a key user that can configure the PMT. He will also manage access rights for PMT users.

5.3. Requirements elicitation technique

To discover the requirements I had discussions and interviews with the coming users of the software. I also clarified how the different tasks where done today. Further, I went through documents that are needed to deliver CBM systems and services. One of the documents was a large Microsoft Excel file that contains the CBM installation base, but also other information about CBM installations and their status, equipment, contracts, and invoicing. Other objects for investigation were the operational tool for CBM services delivery and the Connectivity Service's process and Microsoft Access database.

5.4. Software Requirements Specification

I decided to write the requirements in a document that was based on the IEEE Std 830-1998 (1998) template for software requirements specifications. The Wärtsilä project tools and methods also offered a template for this purpose, which was basically a Microsoft Excel file with ready defined columns for requirements attributes. I concluded that the IEEE template was more appropriate to be used, while its structure was very clear, and also while each of its sections had a specific purpose. In Microsoft Excel the requirements would have been only listed one after another, while in Microsoft Word that I used as editor, I could add diagrams and tables wherever needed. I could also include the use cases in this same document and avoid having the requirements in many separate documents. Finally, I also consider a traditional text document easier to read, because the structure of the document gives a context for the requirements.

1. Introduction
   1.1. Purpose
   1.2. Scope and vision
   1.3. Definition, acronyms, and abbreviations
   1.4. References
   1.5. Document overview
2. Overall description
   2.1. Product perspective
   2.2. Product features
   2.3. User classes and characteristics
   2.4. Operating environment
   2.5. Design and implementation constraints
   2.6. Assumptions and dependencies
3. Functional requirements
   3.1. Actors
   3.2. Actor-Goal list
   3.3. Use cases
4. External interface requirements
   4.1. System interfaces
   4.2. User interfaces
   4.3. Hardware interfaces
   4.4. Software interfaces
   4.5. Communication interfaces
5. Other non-functional requirements
   5.1. Performance requirements
   5.2. Security requirements
   5.3. Operational requirements
   5.4. Other constraints

Figure 15. Table of contents for the SRS.

The IEEE Standard 830-1998 (1998) template is very comprehensive and thus I had to adopt it to fit into this project, see figure 15. I removed many subsections that I found unnecessary. In addition, I split up the Specific Requirements chapter to three separate

chapters, because I wanted to keep the functional requirements that included the use cases separately from the other requirements. Also, the external interface requirements and other non-functional requirements got their own chapters. In some cases I also left the headings in the SRS even if I knew that there will not be any requirements related to it. Instead I only stated that no requirements identified. Last, I included a data dictionary and data model in the appendix of the SRS.

To make every requirement identifiable, I assigned an id to each of them. I named this requirement id based on what the requirement could be associated to, and I also added a number to it. E.g. the user interface requirements got the ids UI-1, UI-2, etc. If some of these requirements were strongly related to each other, then I added an extension number to them e.g. UI-2.1, UI-2.2, etc. In this manner it was easy to group related requirements.

To improve the usability of the SRS, I used hyperlinks to make the navigation within the document easy. Specially, because the document grew in length and because there were many use cases, I decided to connect the use cases listed in the actor-goal list with the actual use case descriptions. I implemented this by adding a hyperlink in the use case title, and in this way it was possible to quickly find the corresponding use case description. I also added a hyperlink in each of the use case description in the title "Use Case ID:" so by clicking on the text would take the reader back to the actor-goal list. Additionally, hyperlinks were practical to use when a use case was included or extended by another use case.

5.5. Functional requirements

I decided to document the functional requirements as use cases. The advantage with the use cases are that they are structured narratives that all the stakeholders can easily understand. Additionally, the use cases clearly state what the user wants to accomplish with the new software, and hence it is also easy then later to verify if the ready software meets the requirements.

The process to define use cases was to first brainstorm primary actors, to find every human and non-human actor over the entire life of the new system. The next step was to

brainstorm the goals that the primary actors want to achieve with the production management software.

I listed the primary actors in actor profile table, see table 9. The purpose of this table was to give the readers an understanding of actor's background and skills that could impact on how the software is designed.

Table 9. Actor profile table.

Primary actor	**Profile: Background and skills**
CBM Manager	The CBM manager uses the PMT frequently to plan and report CBM activities. For these purpose he needs to get forecasts of coming CBM installations, reports of monthly CBM activities. He also follows-up PO and invoicing activities.
CBM System Expert	The CBM System Expert is working with the PMT continuously. He depends much on the information in the tool concerning CBM hardware, system delivery planning details, commissioning, and configuration details. Much of the information in the PMT is inserted and maintained by the CBM System Experts. The CBM System Expert is a technically experienced user with skills on automation and information systems.
CBM Expert	The CBM Expert is using the PMT frequently to deliver CBM reports, maintain contacts and other information related to CBM services delivery. The CBM Experts skills are in engine analysis and mechanical engineering, and hence the information management skills can vary very much.
Requestor	The requestor is an occasional user that uses the PMT to order CBM systems and services. He is not familiar with the PMT and might be impatient and he just wants to get the work done. No information regarding skills.
SAP user	The SAP user is an occasional user that uses SAP data e.g. through e-tools to find which installations are connected to CBM and which installations have Remote Monitoring (RM). He is also interested to know the running hours of installation's equipment. No information regarding skills.

Next, the actor's goals were listed in an actor-goal list, see table 10.

Table 10. Actor-goal list for the production management software.

Use Case	Primary Actor	Task-level Goal	Trigger	Priority
UC-1	Requestor	Order CBM system		
UC-2		Order CBM services		
UC-3	CBM Manager	View report		
UC-4		Configure report		
UC-5		Save report		
UC-6		Get new offers and orders from sales tools	Time	
UC-7		Get contract details from CRM	Time	
UC-8		Get installation and equipment details from SAP	Time	
UC-9	Any	Find CBM installation		
UC-10		View CBM installation status		
UC-11	CBM Expert	Get running hours from Oracle DB	Time	
UC-12		Add contact		
UC-13		Find contact		
UC-14		Update contact		
UC-15		Distribute CBM report		
UC-16		Read CBM report		
UC-17		Upload CBM report		
UC-18		Add comments for a CBM installation		
UC-19	CBM System Expert	View CBM system order		
UC-20		Confirm CBM system order		
UC-21		Complete mobilization details		
UC-22		Complete CBM system details		

In the actor-goal list, I assigned IDs to the use cases to be able to easy refer to them. In addition, I added the columns trigger and priority to the actor-goal list. The trigger column clearly demonstrates which use cases are triggered in some other way than by the primary actor. In this application the only diverging trigger was time. By adding the priority column, it is easy to see which use cases are important and should not be cut from the scope in any circumstance, but also which use cases that can be considered to be excluded or postponed to a later release. Because of time constraints the prioritization of the use cases was left outside this thesis work.

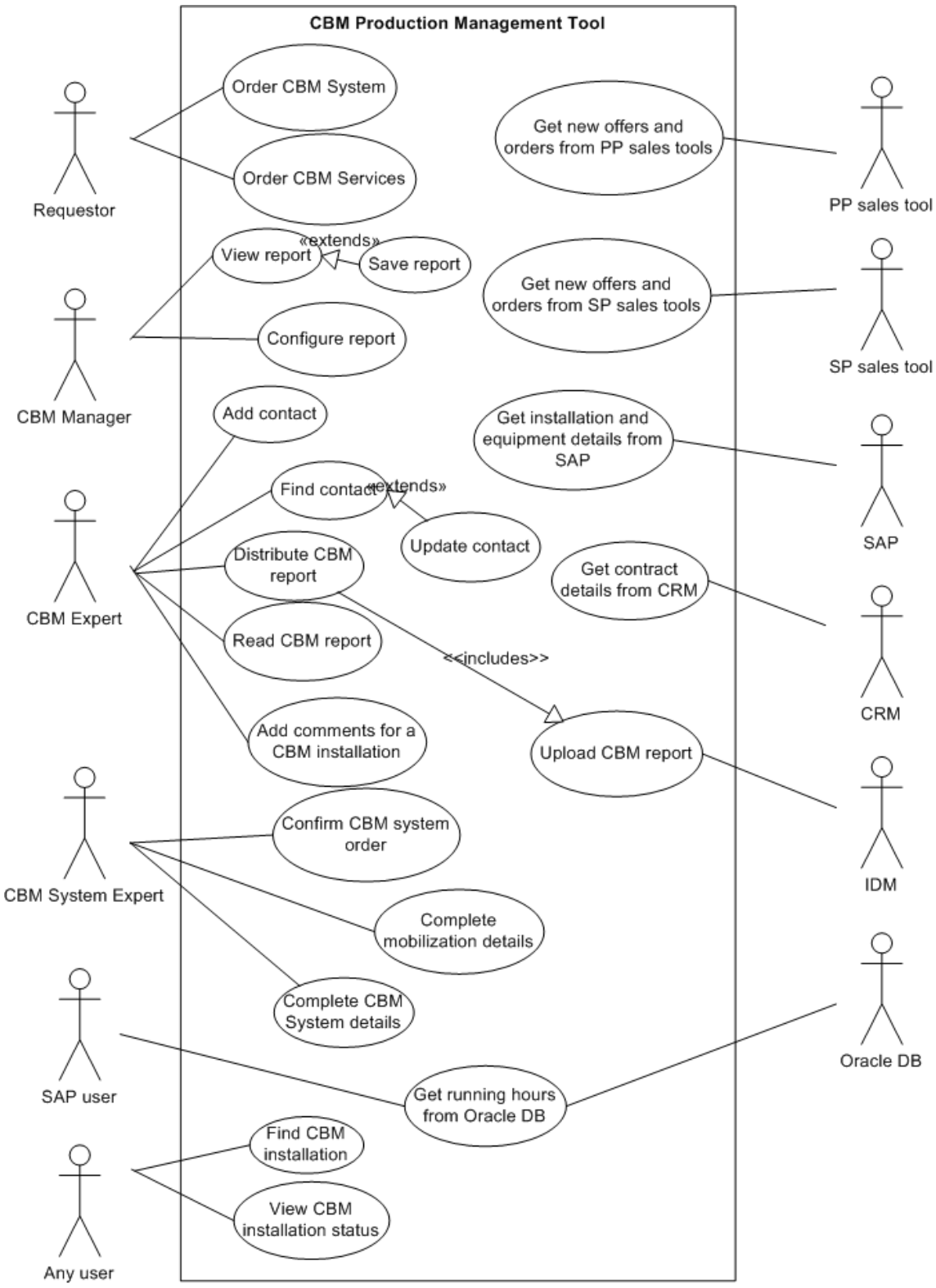

Figure 16. Context level use case diagram.

The context level use case diagram presented in figure 16 is basically a model of the actor-goal list that was presented earlier in this chapter. From this diagram it is easy to note the use cases that are included and extended. These use cases are typically on the sub-function level, while the others are at the user goal level. Further, I put all the primary actors on the left side of the system and the other actors on the right. All the other actors were other systems or applications that the production management tool would interface to.

I documented the use cases in a structured form suitable for a detailed functional requirements specification. The documentation of the use cases proceeded from low precision to higher precision. The actor goal list was the first and lowest level of precision. To increase the precision, I first completed the use case template with the summary that is short description of the use case. I also added the pre- and post-conditions that frame the use case. I preferred to use the terms pre- and post-condition instead of minimal guarantee and success guarantee, because the meaning of these are more obvious to an inexperienced reader of the use cases. Next, I wrote the main success scenario for each use case. At this stage I noticed that some of the use cases were not within the level of user goals and hence had to be reconsidered.

Next, when the use case was properly written at the above mentioned level of precision, I started to brainstorm failure conditions and failure handling. As this is a very time consuming task, this was only done for a few use cases to demonstrate the result. Next, I will present two of the use cases. First, the use case Upload a CBM report and secondly, the use case Distribute a CBM report. These are presented in the tables 11 and 12.

Table 11. Use case: Upload CBM report

Use Case ID:	**UC-17**
Priority:	-
Scope:	Production Management Tool
Level:	Sub-function
Primary actor:	CBM Expert
Summary:	The user uploads a CBM report to the PMT.
Preconditions:	- The user has the CBM report distribution window open.
Triggers:	The user selects to upload the CBM report.
Main success scenario:	1. The user selects to upload the CBM report. 2. The PMT opens a file browsing dialog. 3. The user finds the CBM report on the file system and selects it. 4. The user selects to upload the file. 5. The PMT confirms upload successful. 6. The PMT returns the user to the CBM report distribution window.
Extensions:	TBW
Post conditions:	- The PMT stores the CBM report in IDM under service installations. - The PMT stores the link to the CBM report. - The PMT shows that a file has been uploaded in the CBM report distribution window.
Frequency of Occurrence:	Once every month per installation.
Issues	-

The use case Upload CBM report is a sub-function level use case where the CBM Expert is the primary actor. As depicted in figure 16, this use case is included by the use case Distribute CBM report. Therefore, the main success scenario steps of this use case could replace the calling step in the Distribute CBM report use case. But, I decided to separate these to avoid that there would be too many steps in the calling use case. Also, the precondition and the post conditions are depending on the calling use case. These have to be harmonized so that it is possible to continue the execution of the steps in the calling use case, Distribute CBM report, which is presented in table 12.

Table 12. Use case: Distribute CBM report.

Use Case ID:	**UC-15**
Priority:	-
Scope:	Production Management Tool
Level:	User goal
Primary actor:	CBM Expert
Summary:	The user has completed a CBM report and now needs to distribute it. He uses the PMT to distribute the CBM report to the customer and other stakeholder that are specified in the PMT as a distribution list.
Preconditions:	- The user has logged on. - The user is viewing the dashboard that contains a list of all the installations for which he is responsible.
Triggers:	The user has completed a CBM report and he needs to distribute it.
Main success scenario:	1. The user selects to distribute the CBM report. 2. The PMT opens a CBM distribution window containing the recipients, standard message and signature. 3. The user uploads the CBM report file. 4. The PMT adds a notification that a CBM report file is attached to the message. 5. The user selects the period the CBM report covers. 6. The user sends the CBM report.
Extensions:	*a. The user walks away without notice (time-out): .1 The PMT discards all changes and automatically logout the user after 60min. *b. The user closes or logout from the application: .1 The PMT prompts the user for confirmation. .2 The user confirms. .3 The PMT discards all tasks and logout the user/close the application. 1b. User chooses to use shortcut (only possible if he is CBM responsible person for the installation): .1 The user selects the CBM report distribution link on the dashboard. 2-5a. The user needs to edit the message or signature: .1 The user sets the cursor in the message field and edits the text. 2-5a. The user needs to add or remove recipients: .1 The user sets the cursor in the recipient's field. .2 The user adds or removes recipients by typing or using backspace.

continues…

…continues

	3a. If the user has not been able to make a report, he chooses only to inform the customer that it was not possible: .1 The user edits the message to inform the customer and to state the reason. .2 The user sends the message. 3b. Upload of CBM report fails: .1 The user sends the CBM report manually using his standard mail application. .2 The user marks the CBM report sent in PMT. 4-5a. The user needs to remove the CBM report from distribution: .1 The user selects the CBM report. .2 The user deletes the CBM report. .3 The user returns to step 3.
Post conditions:	- The PMT logs the date the CBM report was sent and set the CBM report delivery status for the CBM report period to "sent". - If no report was attached to the message, the PMT logs the CBM report not sent reason and sets the CBM report delivery status to "not sent". - The user is returned to the place from where he triggered the distribute the CBM report function.
Frequency of Occurrence:	Once every month per installation.
Issues	-

Next, I will explain the content of the different fields of the use case Distribute CBM report, presented in table 12. This is one of the use cases that were finalized, and hence it also includes the extension conditions and the handling of these.

The prioritization of the use cases that would have been entered in the priority field was left out-of-scope for this thesis work. But, this prioritization work would have been one of the tasks that should have been accomplished soon after the use cases were finalized. The prioritization field could also have been omitted in the use case, and then only included in the actor-goal table, see table 10.

All the use cases that I wrote were within the scope of the application itself; the Production Management Tool, which is stated in the scope field. But, the level of the

use cases varied, with a few use cases on the sub-function level, but most of them on the user goal level.

The primary actor of this use case was the CBM Expert, who is referred to as the user throughout the rest of the use case. Next, the summary gives a short description of the contents of the use case. If the use cases would have been written in a casual manner, then all the rest of the fields would have been removed. But, in that case the summary should have been more extensive, than how it is written in the use case Distribute CBM report.

The preconditions and post conditions frame the use case, and therefore I wrote these before the main success scenario. In the precondition I defined in what state the application is before the use case executes and in the post condition how it is left after the execution. In the post condition I also stated possible internal tasks that the application might accomplish as a result of the use case.

The triggers of the use cases were often the same statement as the first step in the main success scenario. Also, this use case would have been perfectly correct with the trigger written in that way.

Next, the main success scenario describes the most likely successful execution of the use case. I aimed on having at least three steps, but at most nine steps in the main success scenario. In cases when there were too many steps, I typically extracted a part of the scenario into an own use case that would then either become a sub-function level use case or another user goal. The use case Upload CBM report, see table 11, was extracted in this way from the Distribute CBM report use case. The link between these two use cases can be noted in step 3 in the main success scenario, where the call for the included use case Upload CBM report is made.

Each step in the main success scenario clearly states a sub goal that is needed to complete the user goal of the use case. In addition, I considered it very important to clearly state in every step who is acting. This was accomplished by always beginning the phrase with “The user…” or “The PMT…”. Further, I noticed also that is was difficult to avoid user interface design in the use cases. Phrases like “The user selects from drop-down list…” or “The User presses the button…” too clearly constrain the later design. But, undoubtedly there could also be circumstances where these are

deliberately meant to be user interface requirements. The user interface requirements will be described in more detail in the next chapter.

In the use case Distribute CBM report I have also specified the extension conditions and the handling of these. As this is a very time consuming task, this was only conducted for a few use cases. First, I brainstormed the extension conditions. Next, I tried to group related conditions that could be handled in similar ways, and also in this way try to avoid too many extensions. Lastly, I wrote the handling of the extension conditions i.e. how these should be handled if the conditions occur. As can be noted from the use case, some of the extension conditions can occur at any step in the use case, while other can occur at one or some specific range of step. In this use case there was only one level of extension conditions and handling of these, but there could also be cases where additional levels are needed. This means that a first level extension handling has some lever level extension conditions that need to be handled.

Lastly, the field frequency of occurrence I also consider important. A use case that occurs frequently should in my opinion get more attention compared with use cases that occur more seldom. Specifically, all the extension conditions and the handling of these should be carefully considered.

5.6. External interface requirements

In the external interface requirements chapter in the SRS I defined the requirements for the interfaces to other business applications and the user interface requirements. Because, I had not involved any experts from the information management department or any experts from the business applications that the PMT would interface to, I only defined these interface requirements on a very general and high-level. Below are two system interface requirements for the interface to the SAP system:

SI-1: SAP
SI-1.1: The PMT shall receive new installations and changes in existing installation data through the SAP application interface.
SI-1.2: The PMT shall receive new equipment and changes in existing equipment data through the SAP application interface.

Next, the following is an extract from the user interface requirements:

UI-2: The PMT dashboard shall list important content and activities that need attention from the current user.
UI-2.1: The PMT dashboard shall show CBM installations that have the status ordered for users in user class CBM System Expert.
UI-2.2: The PMT dashboard shall show a list of all CBM installations for which the logged on CBM Expert is responsible.
UI-2.2.1: Each CBM installation listed on the dashboard shall be a link to the main data for that installation.
UI-2.2.2: Each CBM installation listed on the dashboard shall have a direct link (short cut), to distribute a CBM report.

The user interface requirements could also have been described by pictures or dummy screens. However, then the focus could move more on the layout and graphical aspects instead of the constraints and functionality.

In communication interface requirements I mainly described that the PMT must be able notify users about different things using e-mail messages. For example, the PMT must send a confirmation e-mail to the requestor when a CBM order has been submitted and another e-mail when the CBM order has been confirmed. Another example is that the PMT shall send an e-mail notification containing a request for a new PO to the Contract Manager one month prior to the date when an invoicing period is about to end.

5.7. Other non-functional requirements

In the last section of the SRS I described the performance, security, and operational requirements, but also other constraints that do not have a specific place in the SRS. In the performance requirements I defined measurable requirements for e.g. the time how quickly a search result must execute:

PE-1: The PMT search feature must present the search result within two seconds from the execution command is given.
PE-2: The PMT shall save new data entered by the user within one second from that the save command is given.

The security requirements were related to how the user is authenticated and from where the application can be accessed:

SE-1: All users must be authenticated by the PMT before they can use the application.

SE-2: The PMT shall only be accessible from the company intranet.

Lastly, an example of a constraint was the requirement regarding with which web browsers it should be possible to use the PMT:

CO-1: The PMT shall operate with web browsers that are supported by Sonad computers.

6. DISCUSSION AND CONCLUSION

The aim with this thesis was to discover the requirements needed to develop a production management tool for CBM services. This thesis work was carried out with methods and good practices from the requirements engineering discipline, and the result of it was a Software Requirements Specification (SRS) based on an IEEE Std 830-1998 recommended template. The resulting SRS was not fully completed, but however, it will serve as a good base to start form when also other CBM portfolios are going to be involved in the development of the new software.

A use case approach was selected for both the requirements discovery and requirements documentation. This approach was particularly suitable when stakeholders with little or no knowledge about software engineering are involved in the software development process, but it is also a technique that will find more requirements for a system than many other techniques. The identification of use cases were done by interviewing and having discussions with users and stakeholders. The intension was to understand what the users needed to accomplish with the system, instead of what they wanted the system to do, because user satisfaction can only be achieved by understanding the user needs.

To learn read use cases takes only a few minutes, while learning to write good use cases requires much more effort. To identify use cases was an easy and straight forward task, and all involved stakeholders could contribute. But, to elaborate the use cases required time and much more consideration concerning e.g. the depth of detail and how to fragment them. Especially, the task to write good extensions to the use cases was difficult and time consuming.

There was some work that was left out-of-scope from this thesis, and hence these should be carried out next. First, the requirements were not prioritized. Secondly, the traceability of the requirements should be ensured, e.g. by a matrix that lists the sources and match them to the requirements. Lastly, also similar matrix should be made regarding the dependencies between the requirements.

From the beginning until this point the requirements have been evolving from a simple list to a SRS document written with a word processor. The observation made here is that a clear structure with relevant content makes all the requirements better understood by

the stakeholders. Additionally, a well-structured document also puts the requirements in context. As the work proceeded, the benefits of using use cases to documents the requirements became more and more obvious.

Some of the benefits of well written use cases and requirements emerge in the later phases of the software development. First, these will serve as an input to the functional design. Additionally, the use cases will also serve as an important base for designing the test cases that the new software will have to meet. Finally, they will also be an important starting point when writing the documentation for the new application.

The most import lesson learned was the importance of specifying software requirements that cannot be ignored. The advantages are many. Expensive modification costs in the later phases of the development process can be avoided. The users and software developers can communicate and agree on the functionality of the new software. The software will be verifiable so that it meets the customer needs and understanding. Even though specifying requirements does not seem difficult, it is. However, by applying good practises from the requirements engineering discipline, we have the opportunity to manage in writing good requirements that specify the user needs for the right system. It is vital to build the right system; otherwise it will be a useless system.

REFERENCES

BusinessDictionary.com (2012). Definition of the term Production Management. [online]. [cited 2.10.2012]. Available from World Wide Web: <URL:http://www.businessdictionary.com>

Cockburn, Alistair (2000). *Writing Effective Use Cases.* Boston: Addison-Wesley. 270p. ISBN 0-201-70225-8.

Ghezzi, Carlo, Mehdi, Jazayeri and Dino, Mandrioli (2003). *Fundamentals of Software Engineering, Second Edition*. New Jersey: Prentice Hall. 604p. ISBN 0-13-099183-X.

Haikala, Ilkka & Jukka, Märijärvi (2004). *Ohjelmistotuotanto*. *Kymmenes painos.* Hämeenlinna, Finland: Talentum Media Oy. 440 p. ISBN 952-14-0850-2.

IEEE Std 610.12-1990 (1990). *IEEE Standard Glossary of Software Engineering Terminology*. Standards Coordinating Committee of the Computer Society of the IEEE. 83p.

IEEE Std 830-1998 (1998). *IEEE Recommended Practice for Software Requirements Specifications*. Software Engineering Standards Committee of the IEEE Computer Society. 31p.

Kujala, Sari, Marjo, Kauppinen and Sanna, Rekola (2001). Bridging the Gap between User Needs and User Requirements. In: *Proceedings of the Panhellenic Conference with International Participation in Human-Cumputer Interaction (PC-HCI 2001),* (Patras, Greece, 7–9 December). Greece: Typorama Publication. p. 45-50.

Kulak, Daryl & Eamonn, Guiney (2000). *Use Cases: Requirements in Context*. New York: ACM Press. 329p. ISBN 0-201-65767-8.

Leffingwell, Dean & Don, Widrig (2000). *Managing Software Requirements: A Unified Approach.* Reading, Mass.: Addison-Wesley. 491p. ISBN 0-201-61593-2.

Lilly, Susan (1999). Use Case Pitfalls: Top 10 Problems from Real Projects Using Use Cases. In: *Proceedings of the Technology of Object-Oriented Languages and Systems*. TOOLS 30 Proceedings, Santa Barbara, CA., 1–5 August. Washington: IEEE Computer Society. p. 174–183. ISBN: 0-7695-0278-4.

Maciaszek A. Leszek (2001). *Requirements Analysis and System Design: Developing Information Systems with UML*. London: Pearson Education Limited. 378p. ISBN 0-201-70944-9.

Pressman, Roger S. (2005). *Software Engineering – A Practitioner's Approach*. *8th Edition.* New York: McGraw-Hill. 880p. ISBN 0-07-285318-2.

Rumbaugh, James, Ivar Jacobson and Grady Booch (2004). *The Unified Modeling Language Reference Manual, Second Edition.* Boston: Pearson Education, Inc. 721p. ISBN 0-321-24562-8.

Rumbaugh, James (1994). Getting Started – Using Use Cases to Capture Requirements. *Journal of Object Oriented Programming*. Volume 7, Number 5, September 1994. p. 8–12.

Schneider, Geri & Jason P., Winters (1998). *Applying Use Cases: A Practical Guide.* Reading, Mass.: Addison Wesley Longman Inc. 188p. ISBN 0-201-30981-5.

Sommerville, Ian (2007). *Software Engineering*. *Eighth Edition.* New York: McGraw-Hill. 840p. ISBN 0-07-285318-2.

Sommerville, Ian & Pete, Sawyer (1998). *Requirements Engineering: A good practice guide.* Chichester, England: John Wiley & Sons Ltd. 391p. ISBN 0-471-97444-7.

SWEBOK (2004). *Guide to the Software Engineering Body of Knowledge*. A project of the IEEE Computer Society Professional Practices Committee. 202p.

Vägar, Jens (2012). Discussions with Jens Vägar, Manager Condition Based Maintenance, Wärtsilä Finland Oyj. October 2012.

Wiegers, Karl E. (2003) *Software Requirements: Practical techniques for gathering and managing requirements throughout the product development cycle. Second Edition.* Washington: Microsoft Press. 516p. ISBN 0-7356-1879-8.

Wärtsilä (2007). *Wärtsilä Project Guidelines*: Wärtsilä Operational Development Project Guidelines [online]. Wärtsilä Corporation Ltd. internal document.

Wärtsilä (2009). Condition Monitoring and CBM services. Promotion material.

Wärtsilä (2011). *Wärtsilä Project Model*: Wärtsilä Project Management Guide [online]. Wärtsilä Corporation Ltd. internal document dated 25.11.2011.

Wärtsilä (2012a). Wärtsilä Home Page [online]. [cited 2.10.2012]. Available from World Wide Web: <URL:http://www.wartsila.com>. Wärtsilä Corporation Ltd.

Wärtsilä (2012b). CBM Presentation material. [cited 2.10.2012]. Wärtsilä Corporation Ltd.

Wärtsilä (2012c) Wärtsilä Internal Sales Material, Service Agreements. [cited 2.10.2012].