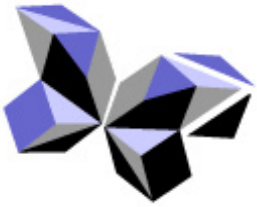


Digital Business Ecosystem

Contract n° 507953

Workpackage 31

Regional catalysts

Addendum to deliverable 31.4

Requirements engineering when collaborating with open source projects

Information Society Technologies

Project funded by the European Community under the “Information Society Technology” Programme

Contract Number: 507953

Project Acronym: DBE

Title: Digital Business Ecosystem

Deliverable N°: 31.4 Addendum

Due date: 30/6/2005

Delivery Date: 11/8/2005

Short Description: Requirements engineering when collaborating with open source projects

Partners owning: Technology Centre Hermia TCH

Partners contributed:

Made available to: All project partners and EC

VERSIONING		
VERSION	DATE	AUTHOR, ORGANISATION
1,0 FINAL VERSION	**30.7.2004**	**JAANA HELIÖ**

Quality check

1st Internal Reviewer: Tuija Kuusisto, TUT

2nd Internal Reviewer:

Executive Summary

In this thesis, requirements engineering collaboration between open source projects and proprietary companies is studied with the Digital Business Ecosystem (DBE) as a case project and seven software producing small and medium sized enterprises (SMEs) as case companies. The aim of the DBE-project is to create an open source infrastructure that can support the evolution and composition of software. The aim of the study is to find out what information flow and information management practices are needed when SMEs are developing software services for the DBE.

Theoretical background of the thesis consists of four parts. In the first part, information management theories are introduced. The role of ontology is examined as well based on software business ecosystem and complex adaptive systems (CAS) approaches. In the second part, requirements engineering is studied. Different workflows and modeling techniques are introduced. In the third part, open source software is examined. Open source software is defined, license matters are discussed, open source community characteristics and the requirements engineering practices of open source projects are described. In the fourth part, the DBE project is introduced. Research approach in the theoretical part of the study is conceptual. In the empirical part, the research approach is action-oriented. Case study method is used for examining seven companies. Interview questions are based on a general recommendation for requirements engineering collaboration that is formulated in the end of part three. Interviews are performed to reach two aims: (1) to understand requirements engineering practices in the companies and (2) to find out information about the knowledge and acceptance of the open source software development characteristics.

The main contribution of this thesis is the description of the four different types of open source software collaboration. Additionally, it was found that information needs and information management practices vary according to the types of collaboration. Drivers possess both will and abilities to collaborate with the DBE, thus, they should be the first to engage in it. Their information needs concern mostly the business opportunities the DBE could offer. Discoverers possess the will but have limited technological abilities; therefore they are interested in the business possibilities of the DBE, not in implementing software services. Implementers possess the abilities but limited will to engage in the DBE. Reorganizing and restructuring of information, better communication and decision-making policies are factors that would increase their interest in the DBE. Users possess both limited abilities and will to participate in the DBE. Their engagement has to be encouraged in the later phases of the DBE with established, ready-to-use business information and technological education.

Table of Contents

TERMS AND ABBREVIATIONS

Active collaboration with the open source community means that the company use of open source software, as part of the company product or in the development of it, is not especially strategic to the company. Collaboration might concern discussing requirements, not implementing software. In general, active collaborators have a positive attitude towards open source software. Actively collaborating companies are quite identical with implementers.

Binary form or executable form of software is compiled from the software source code with programs called compilers. Computers only understand commands in binary form. BML, Business Modeling Language is to be developed in the DBE-project. (More information in chapter 5.3)

BSD, Berkeley Software Distribution, is an open source software license. (More information in chapter 4.2.2)

Business Modeling is a technique for understanding the business processes of an organization. The goal is to specify the relevant business entities to be supported by the system. (More information in chapter 3.1.2)

CIM, Computation Independent Model, models the requirements for the system and describes the situation in which the system will be used. Another term for a model like that could be a domain model or a business model. (More information in chapter 3.2.1)

Community, see open source software community.

DBE-project, Digital Business Ecosystem – project, funded by the EU's sixth framework program. The DBE is a case-project for the thesis representing an open source project (after 2006). (More information in chapter 1.1)

Discoverers are a group of SMEs that possess quite limited abilities but lots of will to co-operate with the DBE-project. (More information in chapter 7)

Drivers are a group of SMEs that possess both the abilities and the will to co-operate with the DBE-project. (More information in chapter 7)

GPL, GNU General Public License: is an open source software license. (More information in chapter 4.2.1)

Implementers are a group of SMEs that possess both the abilities but quite limited will to co-operate with the DBE-project. (More information in chapter 7)

LGPL: Lesser or Library General Public License is an open source software license, which has less strict terms of use than actual GPL license. (More information in chapter 4.2.1)

Linux means actually only the kernel of an open source software operating system, but often the whole operating system is incorrectly referred as Linux.

MDA, Model-Driven Architecture is an approach to system specification that separates the specification of functionality from the specification of the implementation. (More information in chapter 3.2.1)

MIT is an open source software, BSD alike, license. (More information in chapter 4.2.2)

MOF, Meta-Object Facility offers means to express MDA models (CIM, PIM, PSM). (More information in chapter 3.2.2)

Observing collaboration with the open source community indicates that the company uses open source software; typically software development tools made by the community, but rarely gives anything back. The attitude towards open source community might even be a bit negative. Observing collaborator companies are quite identical with users.

OMG, Object Management Group is a non-profit consortium that is committed to developing technically excellent, commercially viable and vendor independent specifications for the software industry. The consortium now includes approximately 800 members. (OMG 2004a)

Ontology is the hierarchical structuring of knowledge about things by subcategorizing them according to their essential qualities.

Open source community or shortly community, is in this study a general term to describe the open source development projects. The community consists of several development projects. (More information in chapter 4.3)

OSI, Open Source Initiative is a non-profit organization, which keeps up a list of the open source software licenses, which meet the terms of their open source definition. (More information in chapter 4.1)

Open source software fills two general conditions: the software source code is available and the user is allowed to modify and redistribute the software quite freely. (See Chapter 4.1)

PIM, Platform Independent Model, is meant for describing the architecture of software excluding details of implementation technology. (More information in chapter 3.2.1)

PSM, Platform Specific Model, is meant for implementing the software solution including details of implementation technology. (More information in chapter 3.2.1)

Proprietary software is owned by a company who produced it. The company alone has access and rights to modify the software source code.

Requirement is a need, condition, or a capability to which a software system must conform.

Requirements engineering includes elicitation, analysis, specification, verification, and management of the software requirements.

Requirements management includes the planning and controlling of activities related to requirements engineering.

SDL, Service Description Language is to be developed in the DBE-project. (More information in chapter 5.3)

SME, small and medium sized enterprise.

Software consists of a program and related documentation.

Software component is a physical part of the program describing its composition. It communicates with other components through one or several interfaces. (Haikala and Märijärvi, 2000)

Source code is written in high-level programming language, which resembles general English quite a lot and in many cases is understandable to humans. Source code of the program is not executable before it is compiled into binary form.

Strategic collaboration with the open source community indicates that the company business idea is somehow based on open source software. Strategic collaboration also indicates that the company is actively involved in the community and understands the basic idea of the open source community: if you take something from the community, it would be advisable to give something back. Strategic collaborators possess enthusiastic attitude towards open source software. Strategic collaborators are quite identical with drivers.

UML, Unified Modeling Language is an OMG standard language for specifying the structure and behavior of systems. It defines an abstract syntax and a graphical concrete syntax.

UMM, Unified Modeling Methodology is a modeling technique, which is concerned only with the inception and elaboration phases of the software development, meaning the most important phases of requirements engineering. UMM is not concerned with the implementation and transition phases.

Use case is each way the user uses the system. Use cases specify sequences of actions, including alternatives of the sequence, which the system can perform, interacting with actors of the system.

Users are a group of SMEs that possess both quite limited abilities and will to co-operate with the DBE. (More information in chapter 7)

1 Introduction

This chapter introduces the reader to the research, Requirements Engineering when Collaborating with Open Source Projects. The background of the research is described first. Next, the research question, objectives and the scope of the study are defined. Finally, the methodology and structure of the study are explained.

1.1 Research Background

In this study the requirements engineering collaboration between open source projects and proprietary companies is observed by using the Digital Business Ecosystem – project and seven software companies as case studies. The history of Digital Business Ecosystem – project lies at the Lisbon summit in March 2000, where the European Union (EU) representatives set the goal of becoming the most dynamic and competitive knowledge-based economy in the world by 2010. Nachira et al. (2002, p. 3) state that the goal was set with the need to promote an "Information Society for All", and the need to address the issues of the digital divide in the adoption of Internet and e-business use. The success of the adoption of digital technologies in Europe is critically dependent on whether the small and medium-sized enterprises (SMEs) are fully engaged in the adoption process. The SMEs are the backbone of the European economy, in most European Union (EU) Member States, SMEs make up over 99 percent of enterprises and do generate a substantial share of Gross Domestic Product (GDP).

Adopting the digital technologies may be an easy job for the SMEs, or on the other hand, not. The SMEs are more flexible in their organisation than larger companies, therefore they may be able to adapt to changing market conditions and new technologies more quickly. However, according to Nachira et al. (2002, p. 3) European small organisations are not ready to use the Internet intensively as a business tool. The survey[1] results point out two main digital divides on e-business:

- While Nordic and Western countries are fast and sophisticated adopters of e-business, the situation is very different in Southern Europe.
- There is a significant difference between SMEs and large companies e-business integration and associated skills.

[1] Eurostat together with the National Statistics Institutes conducted the survey in the period November 2000 and June 2001. The survey covered SMEs with 10-249 employees in 13 EU Member States plus Norway. It reports e-commerce and ICT usage of enterprises in all sectors of the economy.

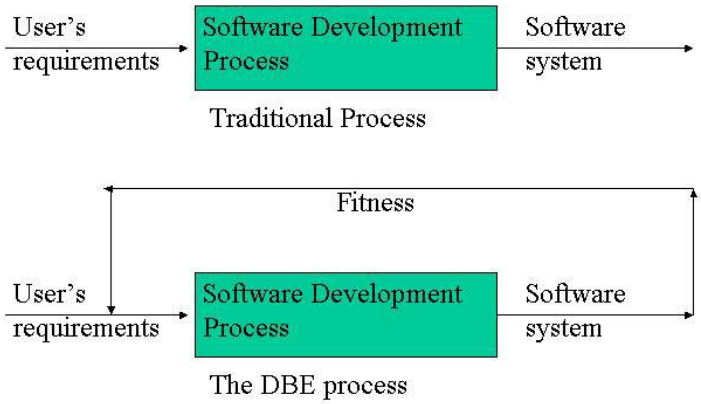
One of the EU's sixth framework program projects is the Digital Business Ecosystem (DBE) project. It aims to provide Europe with innovative software development and recognised advantage by its SMEs and to achieve greater information and communication technology (ICT) adoption in general. In other words, the DBE aims to remove the obstacles SMEs face as they engage in e-business. Those obstacles are lack of knowledge and skills, lack of technological solutions, shortage of capital and complexity of regulations (Nachira et al. 2002, p. 5). The objective of the DBE project is to develop an open source software distributed environment that can support the evolution and composition of (not necessarily open source) software services, components and applications.

What is open source software then? Before defining it thoroughly, explaining a few other terms might be useful. Software consists of a program and related documentation. Software component, then again, is defined to be the physical part of the program, which describes its composition. A software component communicates with other components through one or several interfaces. For instance, libraries, executable programs or source code files can be described as software components. (Haikala and Märijärvi 2000, p. 410) Software source code means functions and directions of a program that are written with a high-level programming language. The source code of a program is quite understandable to humans, since most high-level programming languages today look a lot like general English. Before a program can be executed, the source code of a program has to be compiled into a binary form.

The source code of a program has traditionally been of special value to the software producing companies. The source codes have been available to nobody but the owning company, which is from where the term closed source software originates. The producing company alone has had access and rights to modify the source code, which makes the software proprietary. According to Parviainen (2004), the point in open source software is that source code is accessible to everyone. It is also allowed to make changes to the program and compile it to one's own computer.

The DBE infrastructure is an environment, where the SMEs may declare they have a service to offer and declare their need for a service, too. The declaring is done by a Service Manifest which is described with specific languages, the Business Modelling Language (BML) and the Service Description Language (SDL). The DBE also has a automatic mechanism for receiving feedback of the fitness between the service request and the implementation of the service, which traditionally has been a question of activity of the responsible people. If there are several implementations of the same service request, the implementations are all evaluated. Thus, the DBE has its own mechanisms for

requirements engineering process, for example eliciting, analysing, evaluating and evolving information. See picture 1 for traditional and DBE software processes.



Picture 1. Software development processes.

The requirements engineering mechanisms and processes in the DBE are still to be specified, as the project is only in the beginning. Especially requirements engineering in open source community and with open source community are what this study examines. Requirements engineering includes elicitation, analysis, specification, verification, and management of the software requirements, with software requirements management being the planning and controlling of all these related activities (Dorfman & Thayer 1997). A requirement is a condition or capability to which a system must conform (Kruchten 2000, s.156).

A requirement is data, information, knowledge or a combination of one or more of those depending on a situation. Dixon (2000) defines that data are unsorted bits of fact. Data becomes information when it has been sorted, analyzed and displayed. When people make meaningful links in their minds between information and its application in a specific setting, information turns into knowledge. When applying the above definitions to requirements engineering, one could say, that when a software engineer has analyzed data, the data has turned into information. When the requirement and its setting is completely understood by the software engineer in a way that enables him to implement it into software, the information has turned into knowledge.

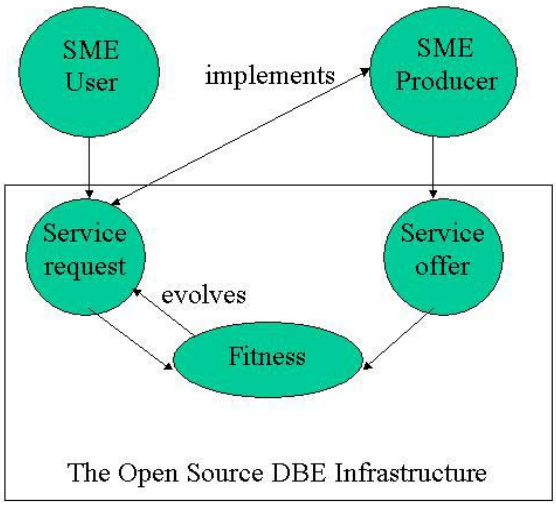
Refining data into information and information into knowledge is not enough when designing and implementing large, multinational and multidisciplinary

software projects, like the DBE. The project has to have a common language, meaning more than English. The project needs to structure knowledge hierarchically and define common meanings to terms. This is where ontology could be of help. Howe (1997) states that ontology has its origin in philosophy, where it means a systematic account of existence. Artificial Intelligence (AI) - environment has its own definition that descends from philosophy: ontology is an explicit formal specification of how to represent the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships that hold among them. An extension of these senses of ontology is the definition for information sciences: ontology is the hierarchical structuring of knowledge about things by subcategorizing them according to their essential qualities.

1.2 Research Problem, Point of View and Definition of the Topic

The research problem of this study is: what information flow and information management practices are needed when the small and medium sized software enterprises are developing software services for the DBE? The thesis is written from the software intensive SMEs point of view in order to let them provide the DBE-project with information that will ease their involvement in the DBE. Another perspective is that of the DBE-project, since the information provided by the thesis can be essential when creating the DBE possibilities for the SMEs.

The information given by the SME user companies is refined in a requirements engineering process into software requirements, which together compose a service request in the DBE. The software producing SME notices the service request and implements the requested service to be offered. The offered service can then be used in the DBE environment and the using experience can be compared to the requested service with a fitness mechanism the DBE presents. The fitness mechanism provides new information that tells how well the implemented service meets the requirements. With that information, new requirements rise and old requirements evolve. This is especially the case if there are several implementations of the same service request from competing SME producers. See the research setting in picture 2.



Picture 2. Research setting.

The requirements engineering process in an open-source software community is different from that of the proprietary software. When developing proprietary software the customer needs are the starting point of the requirements engineering. In the open source community, value creation is done for the users of the free or open source software, for the developers themselves or for the common good of the society (Heikinheimo 2003b, s. 52), so the starting points of the requirements engineering process are quite different. Since the DBE infrastructure is based on open-source software and the software services, both proprietary and open source, will be composed on top of that infrastructure by the SMEs, it is important to define the requirements engineering process in the two-worlds-combining DBE.

The information flow and information management practices are only examined in business-to-business-relationships; individual consumers are not a concern of this study. The open source community and especially different open source licenses are studied only briefly, since the main concern of this study regarding open source community is the collaboration with it. The DBE is a complex technical system but technologies are not the point in this research, therefore, the DBE will be presented on a general level.

1.3 Research objectives

The aim of this thesis is to add to the understanding of requirements engineering process for further specifying the requirements of the DBE requirements engineering environment. In order to reach the objective, the following sub-tasks need to be reached first.

1. Why is ontology needed by the SMEs and how is it used?
2. How do software requirements engineering principles in practice support managing the information needed by the SMEs?
3. What are the ways of working with open source software projects for an SME?
4. What is the requirements engineering recommendation for the DBE?

Software requirements are composed of information, thus, the theoretical part of the research will begin by studying information and different methods of transfiguring it. To be able to answer the research question, information first has to be properly defined and classified. The theory of complex adaptive systems (CAS) will be used as a guideline, as the DBE vision rests on two fundamental ideas: those of self-organization and biological evolution, the same ideas as CAS do (Clippinger 1999, p.1). One of the seven elements of CAS is tagging. As a special case of tagging ontology will be studied to find out, how it can be of help in requirements engineering (sub-objective 1). After studying the experiences of SMEs in the empirical part, the answer to sub-objective 1 can be given.

Requirements engineering process is a part of software engineering process, thus, two different software engineering process models, sequential and iterative, are studied. Requirements engineering and software engineering processes are examined using the Rational Unified Process (RUP) as a guideline. The RUP was chosen because it is based on commercially proven software best practices[1] that are commonly used in industry by successful organizations (Kruchten 2000, p. 5-6). Furthermore, the RUP uses widely accepted Unified Modeling Language (UML) for modeling requirements. The different requirements engineering techniques and procedures of transfiguring information (sub-objective 2) are studied both theoretically and empirically, to answer the sub-objective 2.

The open source community and the ways of working with it (sub-objective 3) are studied to find acceptable characteristics of open source collaboration. A general recommendation of collaboration is presented at the end of open source software theoretical part. The sub-objective 3 is answered in the empirical part after studying the experiences of the SMEs as well.

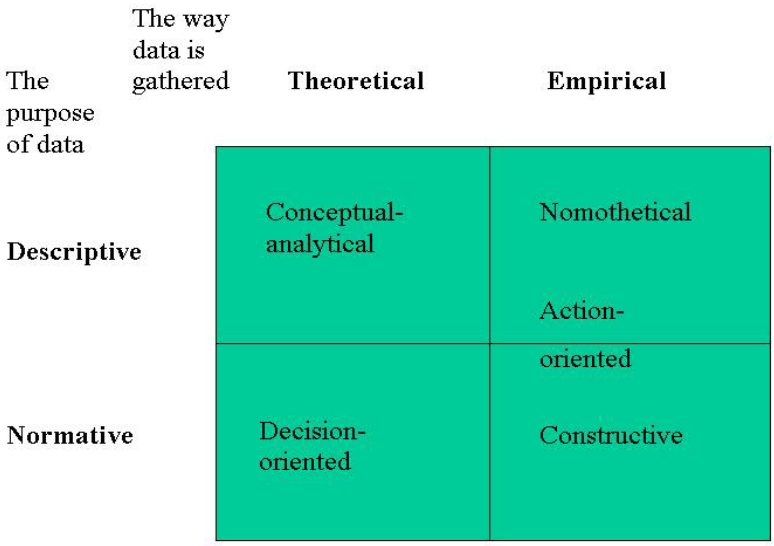
The requirements engineering recommendation for the DBE (sub-objective 4) is composed based on a general recommendation presented in theoretical part, the DBE characteristics, and the results from empirical study. First, the DBE

[1] See the Software Program Manager's Network best practices work at http://www.spmn.com

project, its infrastructure and the pilot applications are introduced Then, the general requirements engineering recommendation and the theory about ontology are utilized to elicit the requirements of software producing SMEs for the open source collaboration. After analyzing the SME answers, the sub-objective 4 can be answered.

1.4 Research approach and method

During the last decades, a research approach classification of Neilimo & Näsi (1980) has established itself. Their classification divides approaches into conceptual-analytical, nomothetical, decision-oriented and action-oriented. Kasanen et al. (1991) have added a constructive approach to this classification, which is situated near the decision-oriented or the action-oriented approach depending on the research method. See picture 3 for the classification of approaches.



Picture 3. Research approaches (adapted from Kasanen et al. 1991, p. 302).

According to Kasanen et al. (1991, p. 302), the classification categorizes approaches according to the purpose of data and the way it is gathered. The purpose of data may be either descriptive or normative and data may be gathered theoretically and/or empirically. Descriptive research aims to explain a phenomenon while normative research aims to find results, which could be used as a guideline when developing operations. Theoretical research aims to develop new theories based on already verified older theories. Empirical research searches for statistically verifiable phenomena among individual cases.

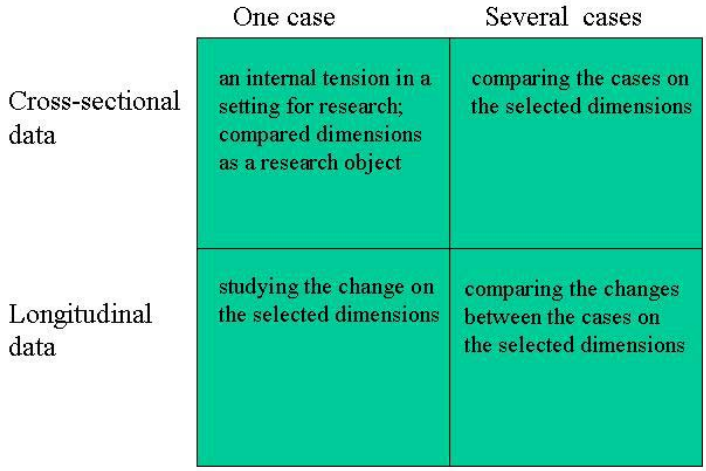
It is typical in the business economics research, that no one approach covers the research alone. The different approaches can be used together and when deciding which is the main approach in a study, the deciding factor is where the acquiring of the new information is situated. Typically, the beginning of the research is conceptual-analytical, and in the end, the empirical data is then gathered and analyzed using that theoretical framework. (Olkkonen 1994, p. 80) That is also the case in this research. First, theoretical recommendation is drafted based on existing theory, and then it is applied in a number of empirical cases.

The research approach in this thesis is both conceptual-analytical and action-analytical. In the beginning of the research when explaining the theoretical background of the study, the data is gathered theoretically and its purpose is descriptive, therefore, the approach is conceptual-analytical. The approach changes when analyzing the empirical data gathered from the SMEs. At that time, the purpose of the empirical data will be descriptive when the interview results are explained. On the other hand, the purpose of the empirically gathered data will be normative when using it to further develop the requirements engineering collaboration recommendation. Therefore, the action-analytical approach is used.

The research method for this thesis is case study. According to Aaltio-Marjosola (2001), research cases are unique, and they are studied in their own special setting. It is important to connect the research setting to the previous theories, which compose the foundation for the analysis and interpretation in the conclusions. In this research, the work for the objectives begins with studying the theoretical background. Interviews, especially the methodologies of a thematic interview, are the most common method for gathering data in a case study (Aaltio-Marjosola 2001). It is typical for the thematic interview that the discussion addresses certain themes, which are known ahead. The methodology lacks accurate format and order of questions that are characteristics of a structured interview. Those are the reasons why thematic interview is described as half-structured methodology. (Hirsjärvi & Hurme 1988, p. 36) In this study, half-structured interviews are chosen for the gathering of empirical data in seven SMEs.

Case data can be either longitudinal or cross-sectional and it may consist of one or several cases. Longitudinal data is used in examining change, the ‘life-circle’ or history of a single unit or comparing the changes between several cases on the selected dimensions. Cross-sectional data is used in examining history, changes on some measurable dimensions, or for example in explaining a phenomenon, such as economic returns, with its internal features. If there are several studied cases then the setting for research can be comparing these

units on selected dimensions. (Aaltio-Marjosola 2001) See the application possibilities of a case study in picture 4.



Picture 4. Application possibilities of a case study (adapted from Aaltio-Marjosola 2001).

According to Aaltio-Marjosola (2001) finding its roots in the theoretical frame is a special challenge for a case study: a clear conceptual frame forms a foundation for interpreting the results of a case study. Therefore, cases in this research are compared on the selected areas of importance that are found while studying the theory. Thus, the data of several cases is in cross-sectional use.

There are different approaches in carrying out the interviews. If the data consists of several cases, a winding approach, which moves from one case to another, may be the best one. In this approach, the researcher gathers data from one case, interprets it and asks new questions based on the interpretation, and then moves on to another case in order to find answers to these questions or to deepen his interpretations. The cycle can be repeated several times until the essential questions of the study have been answered. A process-like progressing approach is different. In it, a set of the studied units is formed in the beginning of the research and data is gathered despite the results of the different cases. In both approaches, the study cases are chosen suitably, not incidentally. The study cases are examined as unique and the data is interpreted accordingly. (Aaltio-Marjosola 2001) This research uses the process-like approach.

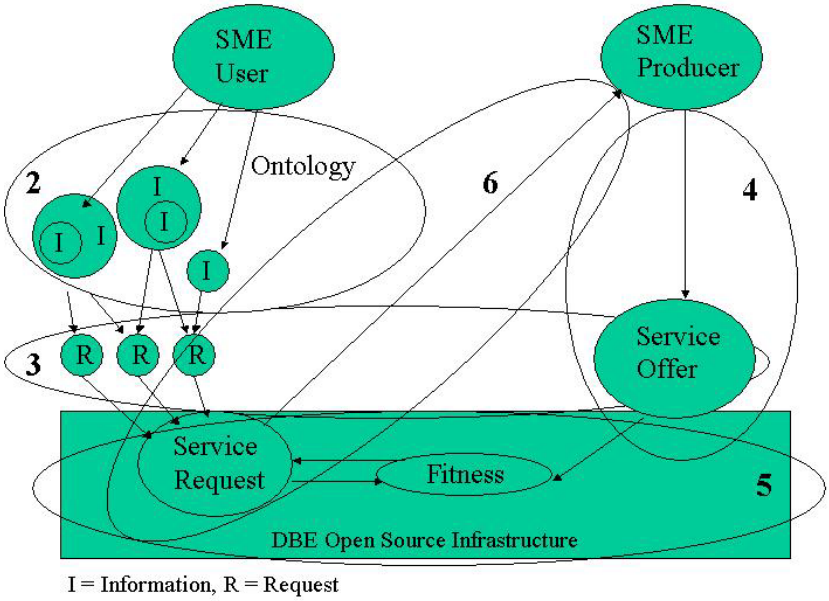
The analysis of the research data, as well as the data gathering, is done in an iterative manner. While the data is being gathered, it is also evaluated, and

therefore, a partially theoretical construction is done during the gathering stage. (Aaltio-Marjosola 2001) According to Yin (1989) firstly in data processing, the data is outlined around themes or questions, and secondly the data's suitability for the categories is examined more thoroughly. In this examination, statistics, tables and figures, which are used for summarizing the data, may be helpful. The categories may change after the analysis, once it is realized that the results require different kind of interpretation. An individual case, for example, might not fit into any of the categories, and cause the re-interpretation of the whole data and the categories.

Aaltio-Marjosola (2001) states that in the continuation of the data processing, the data is further summarized, and interpreted by the researcher. Contextuality of a case data is an essential foundation for interpretation. The data is interpreted with an aim to understand the individual case through the elements found in its own environment, whether they are economic, cultural or social ones. A theoretical frame and a strong conceptual foundation are especially important supporters of an analysis.

1.5 The Structure of the Research

This research consists of introduction, theoretical part, empirical part and conclusions. In the introduction the background of the research and the objectives of the research are explained. Theoretical part consists of three chapters that are needed to explain the underlying phenomenon. First in chapter 2, information and ontology are introduced by explaining what they are, how information can be classified and how ontology is used in the software business ecosystem. Then in chapter 3, the requirements engineering principles and how they can support describing software are explained. Then, the open source software community and its characteristics regarding working with proprietary software enterprises are explained and a requirements engineering recommendation is proposed in chapter 4. The chapters mentioned are corresponding to the numbers of the chapters in the picture 5 and to the numbers of the chapters in this document.



Picture 5. The structure of the research.

In the beginning of empirical part in chapter 5, the Digital Business Ecosystem project, its infrastructure and pilot applications are introduced The empirical part continues with company cases with which the requirements engineering recommendation is tested. The results of SME interviews are presented in chapter 6. Finally, the research is discussed and concluded in chapter 7 by answering the main research problem and assessing the work and its results.

Before beginning the theoretical part, the font styles used in this study are explained. Italics is used to indicate a term or a concept that is being defined nearby. Bold is used to indicate a list item. Finally, text is underlined when there is a need to emphasize a certain word in order for the actual meaning of a sentence to be adduced.

2 Information in Software Business Ecosystem

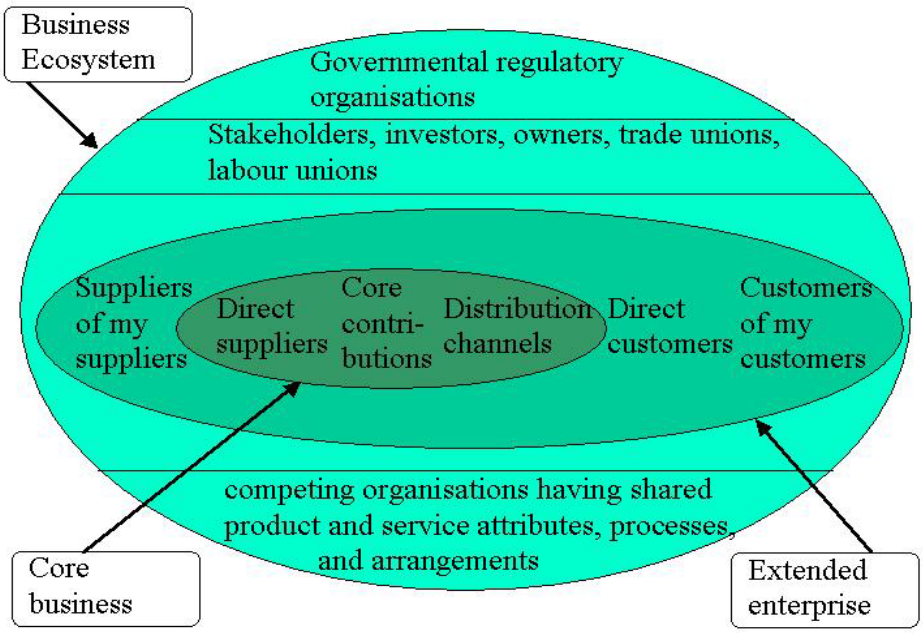
This chapter introduces theories that explain how information is classified and managed in software business ecosystem. First, the concepts business ecosystem and software business ecosystem are discussed. Second, a model for classification of information is explained. Finally, three partially different perspectives for transfiguration of information are given. The chapter introduces a process model for information management of Choo (1998), a theory of complex adaptive systems (CAS) (e.g. Holland 1995, Clippinger 1999), and ontology (e.g. Gruber 1993, Guarino 1998). After given the theoretical background information to the reader, the role of ontology in software business ecosystem is shortly discussed.

2.1 Software Business Ecosystem

2.1.1 Business Ecosystem

Moore (1996, p. 26) describes a Business Ecosystem as an economic community supported by a foundation of interacting organizations and individuals. This economic community produces goods and services of value to customers, who themselves are members of the ecosystem. Over time, they co-evolve their capabilities and roles, and tend to align themselves with the future directions set by one or more central companies. The community values the ecosystem leader, because it enables moving towards shared visions to align their investments and to find mutually supportive roles.

The above definition contains three main points, which for a few years back were rather surprising when introduced to enterprise executives. Firstly, it states that a business ecosystem is a community where different organizations, producers and customers are working together. Co-operation is nothing new, true, but when the concept is broadened to include even competitors and possible competitors (Moore 1996, p. 27), the situation is different. See picture 6 for the business ecosystem and the classification into different categories, based on how close the business relationship is. The enterprise itself and its closest suppliers and distributors compose the core business. Extended enterprise constitutes of the core business, customers, the second-generation suppliers, and other partners. The business ecosystem then is composed of the extended enterprise, competitors, stakeholders and government agencies.



Picture 6. Typical Business Ecosystem (adapted from Moore 1996, p. 27).

Secondly, Moore's definition of business ecosystem emphasizes the co-evolvement of capabilities and roles of organizations. Organizations are already aiming for mutual benefits by strengthening the key customer and supplier relationships. However, according to Moore (1996, s. 12) the most effective firms are learning to lead the economic co-evolution. Companies like Intel and Wal-Mart, seek out for potential centers of innovation where they can bring powerful benefits for customers and producers alike by orchestrating the contributions of a network of players. Their executives must hasten the coming together of disparate business elements into new economic wholes from which new businesses, new rules of competition and co-operation and new industries can emerge.

Thirdly, Moore's definition of business ecosystem says that along with the ecosystem leader, the whole community gains financial interest when supporting one another. Take Moore's (1996, p. 6) example of ABB Canada, which in 1994 suffered from stagnant sales. The new CEO, Paul Kefalas and his organization looked outward to the business environment asking, who are the major shapers of the future in this region whose success ABB could contribute to. ABB approached leading companies and selected prospects because of their importance in influencing the future, regardless of whether it happened to ABB customers. When a company was willing, the company's and ABB's expert representatives worked together to find creative ways to help the company realize its dream. The results were dramatic. More than dozen major customer-partnering arrangements were established and the sales turned

strongly upward. Since most of the new sales were in the context of long-term partnering agreements, revenues could be expected to continue to increase.

2.1.2 Software Business Ecosystem

Moore stated that in a business ecosystem, different organizations, producers and customers are working together, like a community. In the software industry, according to Messerschmitt & Szyperski (2003, p. 228 - 230) co-operation in the software value chain as well as between the other participants of the industry is more a rule than an exception. A total software solution often includes integrated content from a number of software companies. This creates a need and opportunity for different software companies to form business and co-operative arrangements. On the other hand, competition exists especially within a given architecture at the module level, but then again, interacting software modules are complementary, which again creates an opportunity for co-operation. Direct competition is avoided because competitive pricing is difficult due to high creation costs and low replication costs.

Moore's definition of business ecosystem also emphasized the co-evolvement of capabilities and roles of organizations. In the software industry, one way of co-evolving the capabilities of organizations is the development of industry standards. Messerschmitt & Szyperski (2003, p. 232 - 234) define an industry standard to be a specification that is commonly agreed upon, precisely and completely defined, and well documented so that every supplier is similarly free to implement and use it. The industry standard help coordinate suppliers of complementary products but it is not the only such mechanism. The Application Program Interface (API) enables one-to-many relationship, where one software supplier deliberately creates an opportunity for all the other suppliers to extend its product, without the need for a formal business relationship. Open standards on the other hand allow competition as customers can mix and match subsystems from different suppliers without replacing the whole system.

When looking at the future of software industry, Messerschmitt & Szyperski (2003, p. 355) point out that information appliances are predominantly software-based products whose primary purpose is and remains the capturing, manipulating, and accessing information. Moreover, another trend are embedded software-mediated capabilities within different material products, which benefit from enhanced information processing and control. Consequently, improving different methods of information processing and control remains to be important. As examples of means of information classification and refining, different methods are introduced in the next two chapters.

2.2 Classification of Information

There are several different classifications of information as there are many definitions for the different concepts regarding information too. Most classifications of information include definitions of data, information, and knowledge, therefore, those three are chosen for a closer examination here.

Definitions about data are quite unanimous. In addition to Dixon's (2000) "data are unsorted bits of fact" there are several same kind of definitions: data are unorganized and unprocessed facts (Awad & Ghaziri 2004, p. 36); data are a discrete, objective facts about events (Davenport & Prusak 2000). The characteristics of data seems to be that it is in its natural form, data are not sorted, organized and processed. Another common feature in these definitions is that they all state that data are facts; therefore, data are something that has actual existence, like numbers. Awad & Ghaziri (2004, p. 36) give an example: the data could be a number of socks and the price paid for them in a warehouse. However, the data does not tell anything about the motivation of the purchase, the quality of the socks or the reputation of the warehouse. On the other hand, when stores collect data, in time they will be able to evaluate patterns of purchases, number of customers purchasing specific items and other items those customers purchased. Evaluations such as these can be used to derive information about customer behavior, price-sensitivity of certain merchandise and the like. In other words, data is a prerequisite to information.

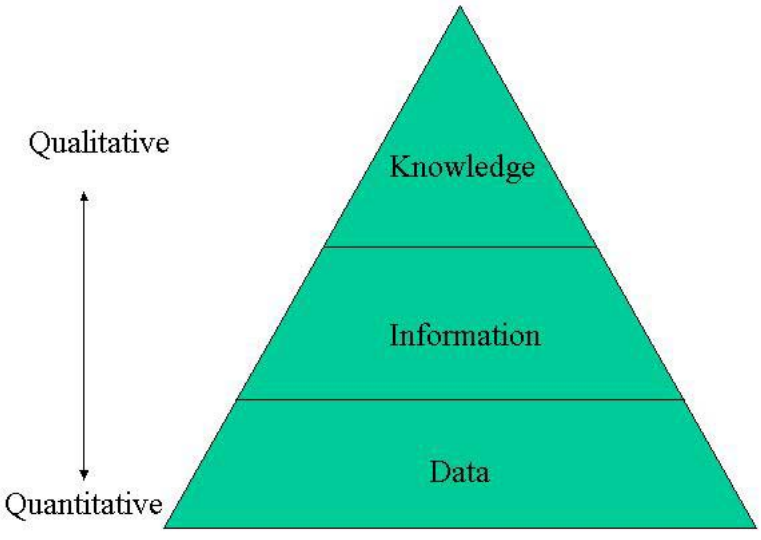
Data becomes information after it is organized and systematized (Acharya 2000). Dixon (2000) thinks quite the same, as she states that information is data that has been sorted analyzed and displayed. Davenport & Prusak (2000) take the definition one step further: they say information is data that makes a difference. Awad & Ghaziri (2004, p. 36 - 37) combine the above definitions by defining: "information means shaping the data to arrive at a meaning in the eyes of the perceiver." The data may have been reorganized, analyzed, have errors removed – all performed to add meaning for instance, to a message or a document.

Knowledge is derived from information like information is derived from data. It may be described as an understanding of information. Knowledge embraces a wider sphere than information; knowledge includes perception, skills training, common sense and experience. The sum of our perceptive processes helps us to draw meaningful conclusions. (Awad & Ghaziri 2004, p. 37) Therefore, it seems that developing information into knowledge definitely means involving humans, their skills and experiences. That conclusion can also be drawn from Dixon (2000): knowledge is meaningful links people make in their minds

between information and its application in action in a specific setting. Acharya (2001) says the same: knowledge involves a human interaction with reality (or with information about reality) where the human is the subject and acts as the active, creative element. Knowledge involves attribution of meaning or significance by the knower as a person.

Defining the term knowledge is reflected by several terms that all denote a particular piece or process in the scope of knowledge (see e.g., Rich 1981a, Prahalad & Hamel 1990, Weick 1995, Grant 1996). Maier (2002, p. 51) lists some examples: ability, attribution, capability, competence, conviction, discovery, estimation, evidence, experience, explanation, finding, hunch, idea, intelligence, interpretation, intuition, know-how, observation, opinion, persuasion, proficiency, proof, sense making, skill, tradition, understanding and wisdom. Thus, it is not surprising that none of the knowledge definitions has succeeded in bringing all these conceptions under one umbrella.

Awad & Ghaziri (2004, p. 40) define the highest level of abstraction to be wisdom, with vision, foresight, and the ability to see beyond the horizon. Wisdom is the summation of one's career experience in a specialized area of work. However, according to Kuusisto (2004) wisdom is part of knowledge. Maier (2002, p. 51) agrees with Kuusisto as he places wisdom in the list of partial definitions of knowledge. See picture 7 for a summary definition of each of these terms.



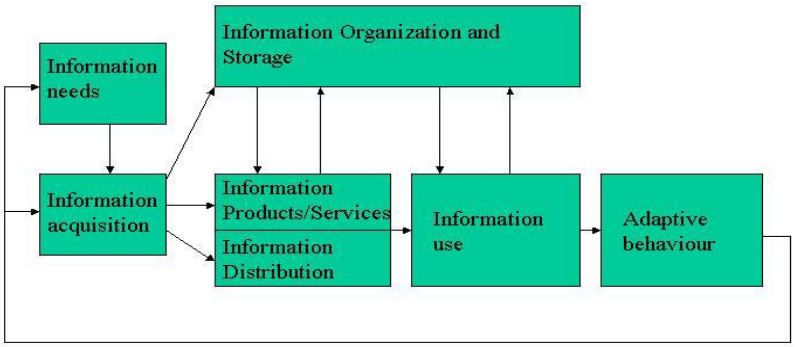
Picture 7. Data, Information and Knowledge.

Picture seven approaches data, information and knowledge in support of the definitions above. Data is produced by the operative systems of a company. It is the raw, quantitative material for data processing. For example in a bank, the massive amount of customers, accounts and account transfers are all data. Information is organized and analyzed data, in that respect it is more qualitative than quantitative than data. When bank customer data is combined with account data, customer accounts are achieved which is meaningful information. When the accounts are combined with account transfer data, customer account transfers are achieved, which is information that is even more useful. Knowledge is a result of people linking information together; it is information that has been refined in people's minds to be taken into action. When an experienced person looks at an account transfer list of a customer, it may, for instance, be evaluated for the ability of the customer to take care of his/her loans.

2.3 Transfiguration of Information

2.3.1 Information Management

Much of the information in an organization is inexact; it is potential more than ready for action. To become strategic, information has to be refined into understanding and knowledge that can guide action. The goal of information management is to transfigure information into learning, insight, and commitment to action. According to Choo (1998, p. 260), information management can be viewed as the management of network of processes that acquire, create, organize, distribute and use information as a continuous cycle. Choo suggests six closely related processes for information management: (1) identification of information needs; (2) information acquisition; (3) information organization and storage; (4) development of information products and services; (5) information distribution; and (6) information use. A process model for information management is presented in picture 8. The most relevant issues in the information management process model for this research are information needs and information acquisition. Therefore, they are explained below in more detail than other issues of the process model.



Picture 8. Process model for information management (adapted from Choo 1998, p.260).

Information needs are discovered from problems, questions, and ambiguities through different situations and experiences in an organization. The information has to be rendered meaningful to certain individuals in certain situations; thus, the meaning of information is not the only thing concerned. Additionally, information conditions, patterns and rules of use should be determined. As information needs are dynamic and multifaceted, only a rich representation of the information use environment enables sufficient specifications. (Choo 1998, p. 262) Taylor (1991) groups the information use environment factors into four categories: sets of people are defined in terms of their information behaviors, which are classified as: the professions, entrepreneurs, special interest groups and special socio-economic groups. Each set of people is concerned with a specific class of problems, which are created by the requirements of its profession, occupation, life-style or interest. These problems will change as the user changes position and perception along with new information. Information behavior is also influenced by four attributes of the work setting: attitude toward information, task domain, information access, and past history and experience. Finally, each set of people has a different idea of what constitutes the resolution of a problem, when the problem can be found resolved. Taylor's framework is a systematic way of analyzing the information requirements of an organization.

Information acquisition sources have to be planned for, monitored, and evaluated just like any other resource of the organization. However, it is a complex task to do. On one hand, the information needs are wide ranging and on the other hand, human attention and capacity are limited. Anyway, the range of sources should be numerous enough and reflect the span of organization's interests. (Choo 1998, p. 262) According to Beer (1974, p. 30), there are two general strategies for an organization to absorb the variety the variety of the

environment and maintain stability: to strengthen variety in the organization and/or to lessen the variety from the environment. Therefore, Choo (1998, p. 263) states that the selection of the information sources should be as numerous and as varied as the external phenomena. Choo (1998, p. 264) continues that the ability to absorb to variety can be improved in many ways: using information professionals, outsourcing specific issues, and using information technology, for example customizable information databases. However, people are the most valuable information sources in any organization. Humans are capable of filtering, summarizing, highlighting and interpreting information, in other words, providing richer communication about an issue. Thus, a plan for collecting and sharing the information by human sources is required.

There are four more processes in the information management process model (picture 8.). First, physical information organization and storing of acquired information is done to facilitate information sharing and retrieval. The requirements for the storage systems are increasing. They are expected to capture different types of information, support multiple user views of data, link related items and allow users to explore patterns and connections. Secondly, information products and services are expected both to give answers to questions and to lead to actions that solve problems. (Choo 1998, p. 264 – 266) Taylor (1986) has identified six categories that enhance information products: ease of use reduces the difficulty in using the product or service; noise reduction excludes unwanted information and at the same time, includes information valuable to the user; quality includes for instance the correctness of information and completeness of coverage on a topic; adaptability means the ability of the service to be responsive to needs of a user in a particular situation and with a particular problem and finally; time and cost savings are the values of service based on the speed of the systems response and the amount of money saved for the users. Thirdly, information distribution is the process by which the right information gets to the right person in the right time, place and format. Finally, information use is the dynamic, social process that makes the meaning, creates the knowledge and selects the patterns of action. Much of its life, organizational information resides in the thoughts, feelings and actions of individuals. Therefore, a high degree of flexibility is required of the information processes and methods when constructing meaning for information. For instance, labeling or naming of concepts and categories has to be relevant to user's interpretations and easy to change. (Choo 1998, p. 267 – 269)

Effective information use results in adaptive behavior. The organization selects and executes pattern of actions that both support the objectives of the organization and which at the same time consider the conditions of the outside environment. The organizations actions interact with the actions of another companies, creating new information to be taken care of, thereby supporting

new cycles of information use. (Choo 1998, p. 261) The adaptive behavior Choo described, has also been studied from the theory of complex adaptive systems (CAS) point of view. According to Clippinger (1999, p. 6), in a complex system, units have no plan concerning how the system should act, and yet the system evolves into a structure adapted to its circumstances. Clippinger (1999, p.5 – 6) states that the behavior of complex adaptive systems arises from the interaction of different subunits with each other, rather than from the behavior of any individual unit or organization, just like Choo above did. Next, the theory of complex adaptive systems and how it is used in managing information is examined.

2.3.2 Self-Organization of Information

This chapter deals with information self-organization and adaptation in an organization from the theory of complex adaptive systems (CAS) point of view. The theory of complex adaptive systems contains a collection of principles and methods that apply across a wide range of sciences – physics, biology, economics, genetics, computer science – and that give deep opinions into how complex systems can evolve from small principles to become well-ordered, adaptive systems (Clippinger 1999, p. 1).

Adaptation is a concept regarding the past and the future of an organization that is widely used with CAS. Clippinger (1999, p. 7) points out that the Darwinian idea of adaptation is a hard-competed world, where the strongest survive, but the CAS perspective on the term is more diversified. Adaptation tells more about the past of the company than it does about the future; a company could be so well adapted to its past that it is unable to see its future. When trying to determine the state of a company at a given time, its capabilities to survive, a more appropriate term would be fitness. Sober (1984, p. 211) agrees and adds that adaptation and fitness are terms, which complement each other. Adaptation describes the past, the history that a characteristic has had. Fitness looks to the future, describing the possibilities of survival and success.

The concept of fitness has been central to evolutionary biology, and it continues to play an important role in CAS too. All complex systems face the same challenge: how to survive in the environment, “fitness landscape”, in which they find themselves. (Clippinger 1999, p. 8) Kauffman (1993) presents that the best fitness, a place to be for a self-organizing system in order to survive, is the place between excessive disorder and excessive order. There, the system is structured enough to act and maintain itself but contains the requisite variety to be maximally responsive to changing environment.

The study of biological evolution, simulations, and artificial life has developed knowledge about what it takes to build self-organizing processes. Seven basic elements are considered the elements of self-organizing systems. They include four properties – aggregation, non-linearity, flows and diversity – and three mechanisms – tagging, internal models, and building blocks. (Clippinger 1999, p. 10) Aggregations are (1) collections of self-organizing entities that (2) in conjunction achieve more than the sum of their individually acting parts (Holland 1995, p. 11). Non-linearity refers to a phenomenon, where a small increment can cause an enormous change (Clippinger 1999, p. 13). Holland (1995, p. 23) points out that non-linear interactions almost always make the behavior of the aggregate more complicated than would be predicted by summing or averaging. Flows can be thought as networks of interactions or nodes, such as people, natural resources, that are not fixed in time. Flows have two important properties: the multiplier effect (a change in a node produces a chain of changes along the way in other nodes) and the recycling effect (recycling in a network with many cycles can increase output a lot). (Holland 1995, p. 23 - 26) Diversity is a measure of variety: the greater the variety, the greater the diversity. Typically, increasing diversity increases the fitness of a complex adaptive system. (Clippinger 1999, p. 15) Holland (1995, p. 27) adds that diversity is neither accidental nor random but it depends on the context. In fact, if a niche-filling content is removed from a system, the system typically responds with a cascade of adaptations resulting in a new content that fills the niche, providing most of the missing interactions. Tagging is a way of naming things to give it certain significance or link it to action. Tagging can be used as a management tool to affect aggregation, flows, diversity, and the fitness of an organization. Tags are critical; definitions should not be too narrow or too broad, both may cause a failure in the market survival. (Clippinger 1999, p. 17) Internal models are simplified representations of the environment that anticipate future actions or events. There are different schools though, arguing to which extent self-organizing systems have internal models. (Clippinger 1999, p. 20) Holland (1995, p. 33) argues that two types of internal models are found in CAS of all kind: tacit and overt. Tacit internal models prescribe current action under implicit prediction of some desired future state. Overt internal models are used as a basis for explicit, but internal, explorations of future alternatives. Building blocks are components that can be combined repeatedly. Building blocks are used to generate internal models; that is an expanding feature of complex adaptive systems. (Clippinger 1999, p. 21). These were the seven basic elements of the self-organizing systems. Next, a closer look on tagging is taken, because as stated earlier, it provides means to affect and manage some of the other seven elements of the self-organizing systems.

Tagging defines what something is and gives it an identity and role in a selection process. Without tagging, natural selection and self-organization are

impossible. Tags launch self-organizing behaviors. (Clippinger 1999, p. 17) Holland (1995, p. 14) adds that tags provide aggregates, which facilitates selective interactions. Tag-based interactions again, are a basis for filtering, specialization, and co-operation. For example a price and brand give a product a certain identity that tempts some customers, others not. They have their impact on the competition as well. Moreover, Holland (1995, p. 13) says that tags are used to manipulate symmetries, which enable us to ignore certain details while directing our attention to others. Clippinger (1999, p. 19 - 20) points out that complex organizations behave differently that simple ones when the business landscape in changing. The past often does not predict the future, because for instance new technologies may redefine the rules of the game. Thus, managers of complex organizations must always look for the new in the familiar; they must critically evaluate, update, and test their internal models of their organizations and their business environments. Managers are also challenged to correctly characterize the environment and to find the right set of tags to define the company value flows. According to Holland (1995, p. 23) defining right tags for company flows is important because tags almost always define the network by delimiting the critical interactions, the major connections. Systems with useful tags spread, while systems with malfunctioning tags cease to exist. Adaptive processes that modify CAS tend to select for useful interactions mediating tags and against malfunction causing tags. With the right set of tags the internal principles of self-organization can be initiated to change, in some cases to be completely transform the organization.

So far, transfiguration of information has been examined with the help of Choo's information process model and from the theory of complex adaptive systems point of view. Tagging was named as the most interesting as it is also useful for managing other elements of the self-organizing systems, for example aggregations and flows. Next, a concept related to tagging is examined; ontology is the hierarchical structuring of knowledge about things by subcategorizing them according to their essential. Ontology has its origin in philosophy, but it is commonly used in research on artificial intelligence and information.

2.3.3 Ontology

Webster's Third New International Dictionary (2002) defines ontology as follows:

1. "(a) A science or study of being: specifically, a branch of metaphysics relating to the nature and relations of being; (b) a particular system according to which problems of the nature of being are investigated

2. A theory concerning the kinds of entities and specifically the kinds of abstract entities that are to be admitted to a language system."

The first sense of the Webster's definition is used in the philosophical tradition, for instance in Albertazzi (1996, p. 1): In contemporary philosophy, formal ontology has been developed in two principal ways. The first approach has been to analyze it using the tools and approach of formal logic as a part of ontology. From this point of view, formal ontology examines the logical features of predication and of the various theories of universals[1]. The use of the specific paradigm of the set theory applied to predication, moreover, conditions its interpretation. The second line of formal ontology analyses the fundamental categories of object, state of affairs, part, whole, and so forth, as well as the relations between parts and the whole and their laws of dependence. The aim is to replace all material concepts by their correlative form concepts relative to the pure "something".

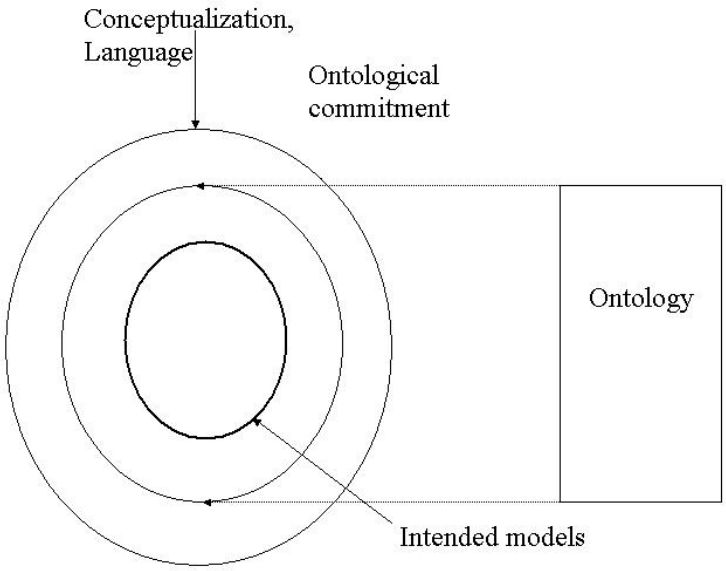
The second sense of the Webster's definition is used in research on Artificial Intelligence and Knowledge Representation, for instance in Gruber (1993): Ontology is a specification of a conceptualization. It has a long history in philosophy, in which it refers to the subject of existence. It is also often confused with epistemology, which is about knowledge and knowing. In the context of knowledge sharing, Gruber uses the term ontology to mean a specification of a conceptualization. That is, ontology is a description of the concepts and relationships that can exist for an agent or a community of agents. Gruber and his colleagues have designed ontologies for enabling knowledge sharing and reuse. In that context, ontology is a specification used for making ontological commitments. For pragmatic reasons, they chose to write ontology as a set of definitions of formal vocabulary. Practically, an ontological commitment is an agreement to use a vocabulary (i.e. ask queries and make assertions) in a way that is consistent (but not complete) with respect to the theory specified by ontology. *Guarino &* Giaretta *(1995, p. 2) have criticized Grubers' definition that it is based on a notion of conceptualization. According to Genesereth & Nilsson (1987), conceptualization is a set of extensional relations describing a particular state of affairs, while the notion Guarino &* Giaretta *have in mind is an intentional one, something like a conceptual grid which they superimpose to various possible state of affairs.*

Before giving his own definition of ontology, Guarino (1998, s. 2) pays attention to some preliminary terminological clarifications. He first considers the distinction between "Ontology" (with the capital "o"), as in the statement

[1] Cocchiarella 1972, 1974, 1986 and 1991.

"Ontology is a fascinating discipline" and "ontology" (with the lowercase "o"), as in the expressions "Aristotle's ontology". The same term has an uncountable reading in the former case, and a countable reading in the latter. While the term “Ontology” seems to be reasonably clear as referring to a particular philosophical discipline, two different senses are assumed for the term “ontology” by the philosophical community and the Artificial Intelligence (AI) community and, in general, the whole computer science community. In the philosophical sense, reference to ontology can be made as a particular system of categories accounting for a certain vision of the world. As such, this system does not depend on a particular language: Aristotle’s ontology is always the same, independent of the language used to describe it. On the other hand, in its most prevalent use in AI, ontology refers to an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words. In the simplest case, ontology describes a hierarchy of concepts and their relationships; in more sophisticated cases, suitable axioms are added in order to express other relationships between concepts and to constrain their intended interpretation. Guarino proposes that the term ontology shall refer to the AI reading, and the word conceptualization shall refer to the philosophical reading. Therefore, while two ontologies may share the same concepts they can be different in the vocabulary used, for instance English or Italian.

Guarino (1998, s. 4-5) *clarifies the role of ontology considered as a set of logical axioms designed to account for the intended meaning of a vocabulary. Guarino makes a clear difference between ontology and conceptualization: “ontology is a logical theory accounting for the intended meaning of a formal vocabulary, that is its ontological commitment to a particular conceptualization of the world. The intended models of a logical language using such a vocabulary are constrained by its ontological commitment. Ontology indirectly reflects this commitment (and the underlying conceptualization) by approximating these intended models.” The relationships between conceptualization, ontological commitment and ontology are illustrated in picture 9.*



Picture 9. Relationships between conceptualization and ontology (adapted from Guarino 1998, p.5).

Guarino (1998, p. 5) stresses that ontology is language-dependent, while conceptualization is language-independent. In its de facto use in AI, the term “ontology” collapses the two aspects, but a clear separation between them becomes essential to address the issues related to ontology sharing, fusion, and translation, which in general imply multiple vocabularies and multiple conceptualizations.

The AI definition of ontology, which refers to a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words, would be the most obvious choice for this study, since the AI definition is used in the computer sciences in general. Guarino’s note that ontology is language-dependent makes that choice a bit difficult though, as this study is a part of the DBE project, which is a multilingual setting. In the DBE, the ontology would be shared among different European countries, which almost all have different languages. On the other hand, at the time of writing this study, no decisions have been made if the DBE ever will use other languages besides English. Therefore, the AI definition can be used in the DBE after all. The objective is to find common understanding of the concepts and their intended meanings. During the time of development, a shared understanding between the development team of the terms involved in the development process can be achieved. However, when the DBE infrastructure is completed and new domains are added in it, the situation is different. Reaching a shared view of concepts between all the domains could be possible, maybe (or maybe not), but extremely time and effort consuming. Therefore, domain specific ontologies would be a more reasonable solution.

2.4 Role of Ontology in a Software Business Ecosystem

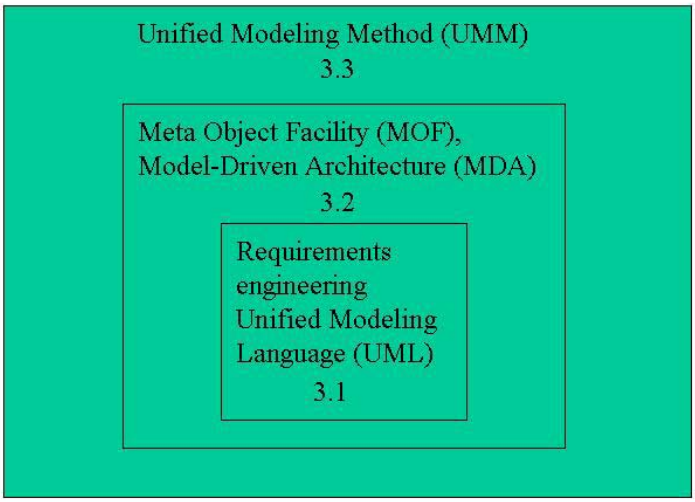
Business ecosystems have a fundamental quality of several organizations working together. Those organizations can be part of the core business value chain or not, they can be even competitors. Through co-operation and co-evolvement of capabilities and roles of organizations, the business ecosystem aims to gain financial interest for all its members.

When operating in a multiple company setting, like in a software business ecosystem, an SME has to consider ontology for several reasons. First, creating ontology facilitates the classification of organizational information. For the greater part, organizational information resides in thoughts, feelings and actions of individuals. Thus, reaching for a common understanding of terms and their relationships through ontology lessens diversity in commonly dealt information. In addition, ontology seems to facilitate co-operation both within and across organizational boundaries. As all parties have at least nearly the same understanding of the mutual concepts and their meanings, the possibility of defects and misunderstandings is a lot smaller. Moreover, the reduction of errors and noise in communication channels facilitates the communication process, saving time and money. Finally, since ontology is closely connected to tagging, it can offer the basis for optimizing organizational change. With the right set of tags, the internal principles of an organization can be initiated to change, as tags provide a basis for selecting patterns of action. On the other hand, an organization with not suitably developed tags may face severe problems. By helping define what is important and what is not, ontology and its associated tags make the process of selecting and evaluating information easier.

So far, information transfiguration has been dealt on a general level. Next, a deeper look into how information is refined into software requirements and modeled with different techniques is taken.

3 Requirements Engineering

In this chapter, the requirements engineering process and different requirements modeling techniques are introduced. The structure of the chapter can be detected in the picture 10.



Picture 10. Requirements modeling methods.

First, the requirements engineering process and the workflows within are examined as a part of software engineering process. Then, Model-Driven Architecture (MDA) is studied along with Meta Object Facility (MOF), which offer means for the requirements modeling to take place at a higher level of abstraction. Finally, the Unified Modeling Method (UMM) is introduced. The UMM is a modeling method that concentrates on software engineering phases most important for the requirements engineering. The UMM can be seen as a process guiding requirements engineering, whether that happens on a higher abstraction level or not. Moreover, although UML is mentioned in the first chapter, it is a modeling language used in all of the above methods and processes.

3.1 Information Modeling in Requirements Engineering Process

Jacobson et al. (1999, s. 24) use a general meaning, need, for the word requirement. Kruchten (2000, s.156) defines a requirement as a condition or capability to which a system must conform. Although these definitions in a simple way say it all, some further explaining of the types of requirements might be useful. Grady (1992) categorizes the necessary attributes of a quality software system as functionality, usability, reliability, performance and

supportability. Although Grady's definition of necessary quality software attributes was already written in 1992, it was not until a few years back when requirements other than functional have been gaining status. One of the reasons for the later observing of, for instance, usability and supportability might be that they are quite difficult to measure and, to a certain extent, matters of personal experience and taste. Kruchten (2000, p. 157) lists some of the considerations for the non-functional quality attributes: usability requirements address human factors like ease of learning and use, consistency in the user interface and documentation. Reliability requirements address frequency and severity of failure, recoverability and accuracy. Performance requirements specify, for instance, the transaction rate, speed, availability, response time and recovery time with which a given action must be performed. Supportability requirements address testability, maintainability and other qualities required to keep the system up-to-date after its release.

Merriam-Webster's Online Dictionary (2004) defines a process as follows: process is a natural phenomenon marked by gradual changes that lead toward a particular result, for instance the process of growth. Process can also be a series of actions or operations conducing to an end; for example a continuous operation or treatment especially in manufacture. The definition of software development process, is compatible especially with the latter one, as Jacobson et al. (1999, p. 24) specify it to be the complete set of activities needed to transform users requirements into a consistent set of artifacts that represent a software product and later, to transform changes in those requirements into a new, consistent set of artifacts.

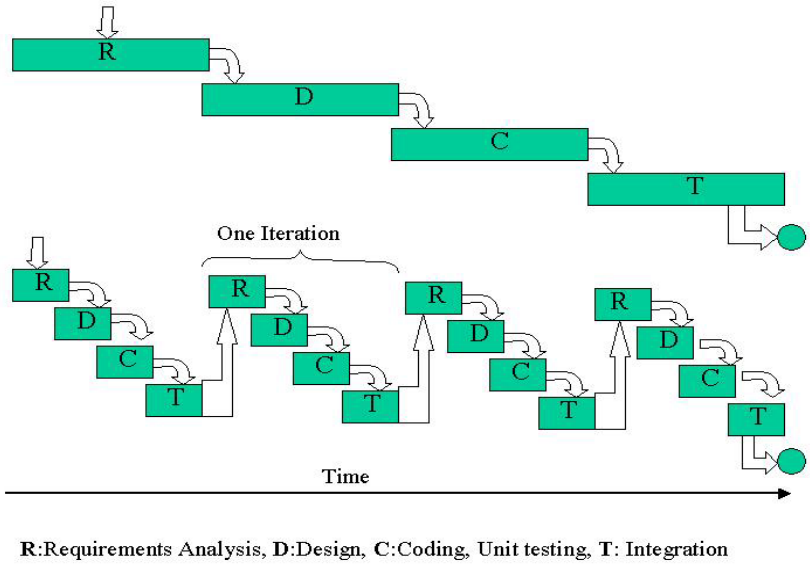
Requirements engineering and requirements management are both an important part of the software development process. The definitions of requirements engineering and requirements managements in literature consist of the same concepts, but are not uniform. Kruchten (2000, p. 25) uses the term requirements management and defines it to be a systematic approach to eliciting, organizing, communicating, and managing the changing requirements of a software-intensive system. Dorfmans & Thayers definition of requirements engineering, on the other hand, includes elicitation, analysis, specification, verification, and management of the software requirements, with software requirements management being the planning and controlling of all these related activities. Thus, if taken only a quick look, the last two definitions seem to be the same, since they both name the same or the same kind of activities, except that Kruchten uses a term requirements managements and Dorfman & Thayer use a term requirements engineering. On the other hand, a deeper observation actually does expose a similarity. Kruchten says "a systematic approach to requirements management" and while writing about requirements management too, Dorfman & Thayer use verbs "planning and controlling". Thus,

if planning and controlling can be described as a systematic approach, the definitions of requirements management by Kruchten and Dorfman & Thayer can be stated to be quite similar. In this study, Dorfmans & Thayers definition is used because of its more accurate definition of requirements engineering.

Managing the requirements of a software system is challenging. It is impossible to completely state the requirements of a system before the start of development. Therefore, identifying the requirements of a system is a continuous process. Furthermore, the requirements are dynamic: they will most likely change during the life of a software project. Indeed, as a new or evolving system changes, a user's understanding of the system's requirements also changes. In the following chapters, the requirements engineering process is described as part of the software engineering process and as itself.

3.1.1 Requirements Engineering in Software Development

Software development has been a sequential process for the past few decades. The sequential process begins with completely understanding the problem to be solved, all of its requirements and constraints. The requirements and constraints are captured in writing and all interested parties are asked to agree on what is needed to achieve. Next, a solution that satisfies all requirements and constraints is designed. Once again all interested parties are asked to examine the design carefully and agree that it is the right solution. Then, the solution is implemented, that is, it is coded with a programming language and tested. Finally, the integrated software components are verified to ensure that the stated requirements were satisfied. After that, the software is ready to be delivered, as can be seen in the upper part of the picture 11 below.



Picture 11. Sequential and iterative software development (adapted from Kruchten 2000, p. 61).

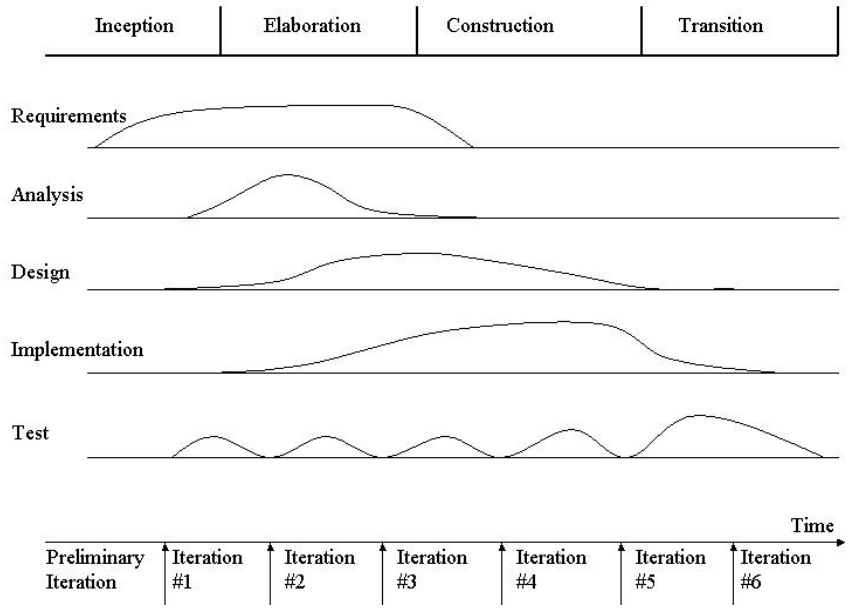
Iterative process is another way of developing software. It can be described as several sequential processes in succession (Kruchten 2000, p. 60). Jacobson et al. (1999, p. 87) say iterative software development means taking small manageable steps: plan a little; specify, design and implement a little; integrate, test and run each iteration a little. The difference between sequential process and iterative process can be seen in picture 11 above.

In the sequential process, it is assumed that the problem, its requirements and constraints, are captured entirely in the beginning. This usually proves to be impossible; requirements will change for many reasons. The users might change their needs, when they see other systems and features and become better educated. The problem can change too. The users do not always know what they want, but when they see it, they know what they do not want. Therefore, trying to capture and freeze the requirements early in the beginning could lead to the delivery of a system that actually does not meet the needs of the users. (Kruchten 2000, p. 55-56) Jacobson et al. (1999, p. 89) add, in a situation like that, changing the system to meet the actual requirements of the users becomes expensive. In addition of managing changing requirements, the sequential process is a risky choice if the system is somewhat larger, complicated, and has novelties in it (Kruchten 2000, p. 57). It would be better to build up the system incrementally and iteratively. The inevitable requirements and other changes are then easier to handle. (Jacobson et al. 1999, p. 89) Kruchten (2000, p. 76) agrees and adds that in the iterative process risks are mitigated earlier. Furthermore, through several requirements refining and testing iterations, the software will have been running longer than with the conventional sequential process. Thus, the iterative process produces software with better quality.

The iterative process is organized in phases inception, elaboration, construction and transition. The four phases constitute an initial development cycle when a software product is created. If the life of the product continues after this point, the product will evolve with the repetition of those four phases. These periods are called evolution cycles. However, these evolution cycles are not the same thing as iteration cycles. Every one of those four phases mentioned, consists of one or several iterations, as seen in the picture 11. (Kruchten 2000, p. 62-64)

In the picture 12, the relative emphasis of various types of activities can also be seen. In the inception phase, the focus is on formulating the scope of the project so that the acceptance criteria for the product can be derived. That is, understanding and capturing the overall requirements and constraints is the most essential activity along with planning the project and its resources and synthesizing candidate architecture. (Kruchten 2000, p. 67) In the elaboration

phase, the focus is on requirements and architecture. The vision is elaborated and a solid understanding is established of the most critical requirements that drive the architectural and planning decisions. Some designing and implementation is aimed at prototyping the architecture. (Kruchten 2000, p. 70-71)



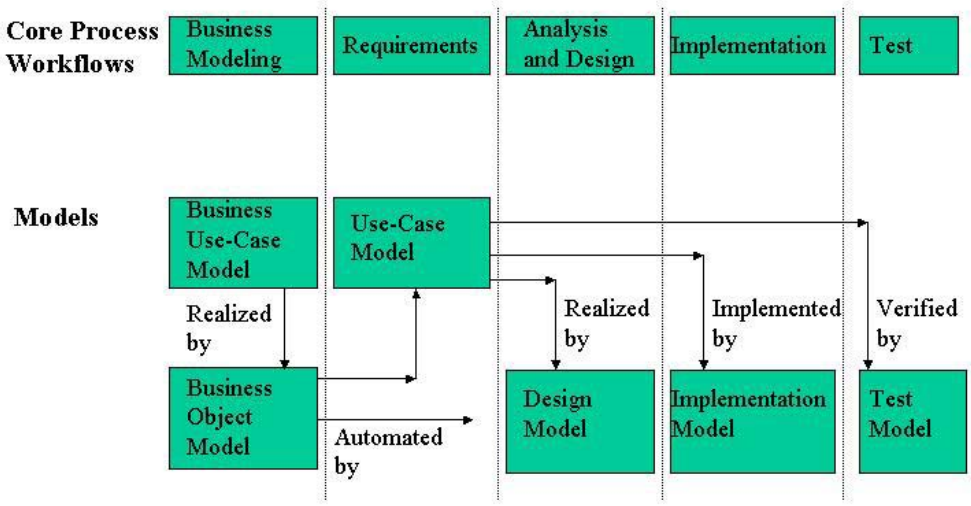
Picture 12. Importance of activities across one development cycle (adapted from Jacobson et al. 1999, p. 118)

In the construction phase, the last adjustments to the requirements are made, but mostly the phase focuses on completing and testing the software components. (Kruchten 2000, p. 72) The transition phase focuses on activities required to get the software into the hands of the users. Those activities include several test-releases, bug fixing, documentation, user training and user support. (Kruchten 2000, p. 74)

Jacobson et al. (1999, p. 34) explain the workflows in the picture 13 (p. 30) by describing the artifacts created in each activity. The Unified Modeling Language (UML)[1] is a widely accepted method for producing the artifacts. The development begins by capturing the requirements as use-cases in the use-case model. The use case model is a model of a system containing actors, use cases and their relationships. (Jacobson et al. 1999, p. 133) It describes what the system does for each type of user. Each type of user, then again, is represented as one or more actors. Actors are the parties outside the system

[1] An Object Management Group (OMG) standard language for specifying the structure and behaviour of systems. It defines an abstract syntax and a graphical concrete syntax. See for example Rumbaugh, J., Jacobson, I. & Booch, G. 1999. The Unified Modeling Language Reference Manual. Addison-Wesley. 550 p.

that collaborate with the system. A use case is each way the actors use the system. Use cases specify sequences of actions, including alternatives of the sequence, which the system can perform, interacting with actors of the system. (Jacobson et al. 1999, p. 134- 135) After creating the use-case model, the system is analyzed and designed to meet the use-cases; thus, an analysis model is created first and a design and deployment model next. The system is then implemented in an implementation model, that is, the code and the components. Finally, the system is verified against the functionality described on the use-cases with a test model. The use-case model is primarily described in natural language, which makes it the least formal of the models mentioned. The implementation model is the most formal, in the sense that parts of it can be machine interpretable. (Jacobson et al. 1999, p. 34)



Picture 13. Use cases ”flow” through the various models (adopted form Kruchten 2000, p. 107)

Use-cases have been adopted for capturing requirements almost universally, but they are more than that. Use cases drive the whole development process and bind together the different workflows. (Jacobson et al. 1999, p. 34) The use case-model is a basis for the design, implementation and test models, as can be seen from the picture 13. Yet, before constructing a use-case model in the requirements workflow, business modeling- workflow is performed and business use-case- and business object- models are constructed. One of the goals of business modeling is to derive the requirements needed to support the target organization (Kruchten 2000, p. 139). Thus, next chapter will have a brief look

at what business modeling is and what it has to do with requirements engineering.

3.1.2 Business Modeling Workflow

Business modeling is a technique for understanding the business processes of an organization. The goal is to specify the relevant business entities to be supported by the system. Accordingly, a business model describes the business processes of a company in terms of business use-cases and business actors. Like a use-case model of a software system, the business use-case model presents the business from the usage perspective and outlines how it provides value to its customers and partners. (Jacobson et al. 1999, p. 115) Jacobson et al. (1999, p. 124-125) continue that a business model is developed in two steps:

- First, a business use-case model, that identifies the use-cases and the actors that use the use-cases, is prepared. With the model, the modelers understand better the value the business provides to its actors.
- Second, a business object model, which consists of workers, business entities and work units, that together realize the business use-cases, is developed. With these different objects, the rules and regulations of business are associated.

A business model is used as a starting point to derive a first set of actors and use-cases for the software system to be built. Thus, every use case of the software system can be traced back to the customers of the business through the actors and business use-cases. (Jacobson et al. 1999, p. 125) Requirements workflow is the next step in the software process. It is examined in more detail next.

3.1.3 Requirements Workflow

The purpose of the requirements engineering workflow is to aim development toward the right system. To achieve this, the system requirements must be described well enough so that the users and the developers can reach an agreement about what the system should and should not do. (Jacobson et al. 1999, s. 113) According to Kruchten (2000, p.160) understanding the user needs is not enough, an additional level of specificity is needed to translate the needs into specification that can be designed and implemented by the developers. Nevertheless, Jacobson et al. (1999, s.113) point out that the requirements are described with the language of the customer (user), because users primarily are non-computer specialists.

Kruchten (2000, p. 163) states that the first step of the requirements workflow is analyzing the problem. This phase includes identifying stakeholders, persons who have a stake in the project outcome, identifying the boundaries and constraints of the system, and gaining agreement on a statement of the problem that is going to be solved. The next step is understanding stakeholder needs and defining the system, which Jacobson et al. (1999, p. 113 – 117) name capturing requirements. There are different starting points for the requirements capturing like there are unique software projects. Jacobson et al. (1999, p. 113) list some starting points: business model, requirements specification or just a vague notion. In between these extremes are all varieties of combinations. Despite the different starting points, Jacobson et al. (1999, p. 114) state that there are certain steps that are feasible in most cases, which allow a suggestion of a workflow archetype. This workflow includes the following steps, which actually are not performed separately:

- Listing candidate requirements
- Understanding the system context
- Capture functional requirements
- Capture non-functional requirements

Listing candidate requirements is what customers, users, developers and analysts do during the life of a system. Candidate requirements are good ideas that some day might turn into a real requirement. This feature list, as it is called, grows when new ideas are added, and shrinks, as features become requirements. The feature list is only used for planning the work. (Jacobson et al. 1999, p. 114) Kruchten (2000, p. 160) agrees and states that each feature might include additional values, such as priority, risk and level of effort and cost, which will help managing the scope of the system.

Understanding the system context is important especially for the key developers of the system. There are at least two ways of expressing the context of a system for the developers in a usable form: domain modeling and business modeling. (Jacobson et al. 1999, p. 115) Business modeling was introduced already in chapter 3.1.2. A domain model is a simplified variant of a business model, according to Jacobson et al. (1999, p. 119), it aims to capture the most important types of objects in the system context. The domain objects denote the “things” that exist or events that occur in the working environment of the system. The domain objects are typically found from

- Business objects, such as orders, accounts and contracts
- Real-world objects, such as enemy aircraft, missiles and trajectory

- Events that will transpire or have transpired, such as aircraft arrival and aircraft departure.

Capturing functional requirements is achieved with the help of use cases. Use cases are a method of describing what each user wants the system to do for him or her. Each user needs several different use cases, each representing a different way he or she uses the system. Consequently, if analysts manage to describe all the use cases that the users need, then they will know what the system is to do. (Jacobson et al. 1999, p. 116)

Capturing non-functional requirements means specifying system properties, such as performance, maintainability, and reliability (explained in chapter 3.1). Some non-functional requirements refer to a real-world phenomenon; thus, they should be captured on the corresponding business or domain object model of the context of the system. (Jacobson et al. 1999, p. 116) Other non-functional requirements cannot be associated with any particular use-case; they may have impact on several use-cases or none at all. Non-functional requirements are for example:

- interface requirements, which specify the interface to an external system or define the set-up in such an interaction,
- physical requirements, which can be used, for instance, to represent hardware requirements; a physical requirement typically specifies characteristics that a system must possess such as its material, shape, size, or weight,
- constraints for the design of a system, such as extensibility and maintainability constraints and
- constraints for the implementation. (Jacobson et al. 1999, p. 128)

The third step in requirements workflow is managing the scope of the system. That involves collecting information linking with requirements to be used in prioritizing and focusing the system. The fourth step is refining the system definition; it includes detailing the requirements to come to an agreement on the functionality of the system. In addition, it captures other requirements, like non-functional requirements, constraints, and so forth. Finally, managing requirements and their changes is performed throughout the workflow. Managing changing requirements means maintaining agreement with the customer and setting realistic expectations on what will be delivered. There, using a central control authority, as Change Control Board (CJB) could be of help. (Kruchten 2000, p. 165) Next, change management is explained further.

3.1.4 Requirements Evolving

So far, the beginning of the requirements lifecycle has been covered. Throughout the project lifecycle, the captured requirements are divided into proper-size slices and detailed in an incremental fashion. As these requirements are detailed, flaws and inconsistencies will undoubtedly be found, which means it is necessary to go back to the stakeholders for further clarification. Thus, the stakeholder needs and requirements are evolving in an iterative manner until all requirements are defined, feedback is considered and the changes are managed.

Kruchten (2000, p. 211 - 212) states that change request management deals with the capture and management of requested changes generated by internal and external stakeholders. A change request is “a documented proposal of a change to change one or more artifacts”. Besides to add a requirement, change requests can be created for many other reasons as well: to fix errors or to improve product quality for instance. Change requests are labeled with states according to the point of lifecycle they are in. Different states could be such as new, logged, approved, assigned and complete. As the change request goes from state to state, certain information has to be added. For example

- the reason and motivation for change,
- the artifacts that are affected and must be modified, and
- the impact on design, architecture, the cost and the schedule

are things to be documented. The change requests have to analyzed and prioritized in order to find out the impact they have on the existing system and documents. Using this analysis the management can decide when the change requests will be implemented and in which release they will be implemented. Noticeable is, that not all requests are acted on. The life cycle of a change request closes when the request has been completed, tested and included in a release.

This chapter concludes describing the requirements engineering process and the different workflows in it. Next, the modeling is taken one step further. Model-Driven Architecture (MDA) is a tool with which the formalization of business logic can be platform- and technology independent.

3.2 Describing Requirements using Model-Driven Architecture

The software code has long been dependent on the implementation technology. Currently there is a shift going on from software design, where the code is

dependent on the implementation technology, to a new paradigm, where business modeling and domain modeling take place at a higher level of abstraction. This new approach is called Model-Driven Architecture (MDA™). (Dini et al. 2003, p. 3)

3.2.1 Model-Driven Architecture (MDA)

Model-Driven Architecture is an approach to system specification that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform. Miller & Mukerji (2003, p. 12) state that MDA is an approach to using models in software development. A model of a system is a description or specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text of which the text may be in a modeling language or in a natural language. MDA concepts are presented in terms of some existing or planned system. That system may include anything: for example a program, a single computer system, some combination of parts of different systems, or a federation of systems. Much of the discussion focuses on software development within the system. MDA provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification; that is what makes it model-driven. MDA development consists of three steps: (1) Developing a Platform-Independent Model; (2) Developing a Platform-Specific Model; and (3) Generating the application (Siegel et al. 2001, p. 4 –8). Miller& Makurji (2003, p. 19) point out that developing a Computation Independent Model is the first step, before the three steps mentioned. Next, a closer look at those steps is taken.

Siegel et al. (2001, p. 4) state that MDA development starts with the construction of a Platform[1]-Independent Model (PIM), which consists of a base PIM and next level PIMs. A PIM is a view of the system from the point of view that hides the details necessary for a particular platform (Miller& Makurji 2003, p. 14 –15). Miller& Makurji (2003, p. 19) argue that developing an MDA model begins with creating a Computation Independent Model (CIM) first, and then a PIM is constructed. The two different statements might be confusing, but in fact, they are not that different after all. Namely, Siegel et al. (2001, p. 4) explain that the base PIM expresses only business functionality and behavior. Business and modeling experts working together build it, as much as possible, undistorted by technology. Miller & Makurji (2003, p. 19) then again, explain that the CIM models the requirements for the system and describes the situation in which the

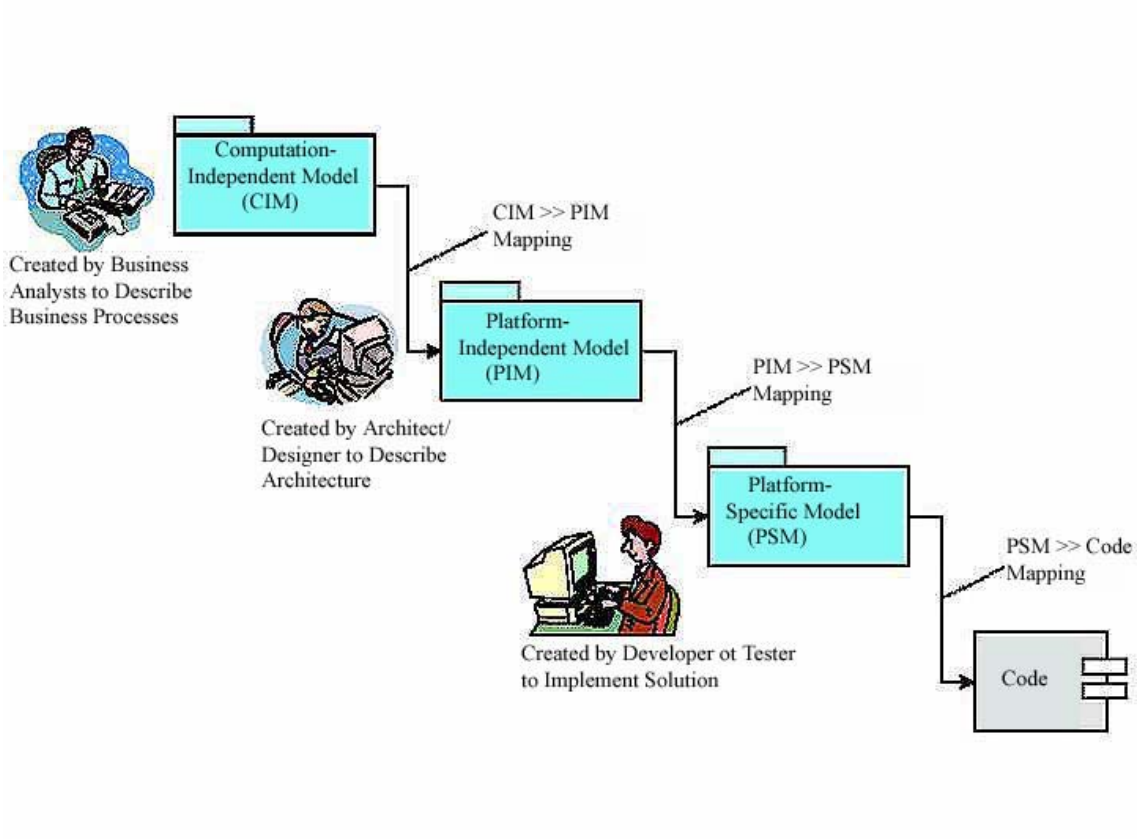
[1] A *platform*, in general, is a set of systems/technologies that provide a set of functionality through interfaces and usage patterns. Any subsystem can use the platform without having to know how the platform is implemented. (Miller & Mukerji 2003, p. 13)

system will be used. Another term for a model like that could be a domain model or a business model (discussed in chapter 3.1). As can be noticed, both Miller & Makurji and Siegel et al. are presenting the same action; only the concept name has changed from the base PIM to CIM, as the MDA modeling technique has evolved for two years from 2001 (Siegel et al.) to 2003 (Miller & Makurji). This research uses the term CIM for a model that concentrates on business functionality. The term PIM is used for a model that is based on a CIM, developed a little further but still has no platform specific details in it.

According to and Miller & Makurji (2003, p. 19) CIM is expressed in UML and Siegel et al. (2001, p. 5) say the same about PIM. The following UML models are used to express the CIM and the PIM:

- class and object diagrams incorporate the structure,
- sequence and activity diagrams embody the behavior,
- class and object names, along with semantic notations, incorporate business factors while
- other aspects of the model incorporate platform-independent aspects of component structure and behavior (Siegel et al. 2001, p. 7).

Siegel et al. (2001, p. 3) explain that UML is a standard way of constructing, viewing, developing, and manipulating an application model at analysis and design time. UML designs can be evaluated and critiqued when changes are easiest and least expensive to make, before the design is coded. Siegel et al. (2001, p. 5) add that the technology-independent UML modeling environment allows business experts to ascertain, that the business functionality embodied in the CIM is complete and correct. Moreover, the technological independence allows the CIM retain its full value over the years, requiring change only when business conditions mandate. Noticeable is that although the PIM includes some aspects of technology, the platform-specific details are still absent. However, adding concepts like persistence, transactionality, security level, and even some configuration information to the PIM, enables it to map more precisely to a Platform- Specific Model (PSM), the next step of the MDA development. The CIMs and the PIMs are created iteratively and they are stored in the Meta-Object Facility (MOF, described in the next chapter) once the first iteration is complete (Siegel et al. 2001, p. 7). See picture 14 for the different models and their relationships.



Picture 14. The creation of MDA systems (adopted from OMG 2004b, p. 1).

The PIM is input to the mapping step, which will produce a Platform-Specific Model (PSM). To produce the PSM, a target platform or platforms for the modules of the application have to be selected. PSM is expressed in UML too, just like the PIM and the CIM. There are specializations and extensions to UML, termed a UML Profile, which give it the power to express CIMs, PIMs and specific platforms. During the mapping step, the application model that in a general level contains the run-time characteristics and configuration information is converted to the specific forms required by the target platform. Automated tools that are guided by an OMG-standard mapping perform as much of this conversion as possible. If there are ambiguous portions found, they are flagged for programming staff to resolve by hand. (Siegel et al. 2001, p. 7) Miller & Makurji (2001, p. 14) list four ways to move from a PIM to a PSM. In increasing level of sophistication and automation, they are:

1. A human could study the PIM and manually construct a PSM, perhaps manually constructing the one-of refinement mapping between the two.
2. A human could study the PIM and utilize models of known refinement patterns to reduce the burden in constructing the PSM and the refinement relation between the two.

3. An algorithm could be applied to the PIM and create a skeleton of the PSM to be manually enhanced by hand, perhaps using some of the same refinement patterns in way number two.
4. An algorithm could create a complete PSM from a complete PIM, explicitly or implicitly recording the refinement relation for use by other automated tools.

Siegel et al. (2001, p. 8) state that generation of software source code from the PSM is the last step of the MDA development. This transformation can be thought in terms of the four levels of sophistication listed above. However, since many development tools already generate interface code from models, evolution here has already passed through the primitive levels 1 and 2. You can expect even early MDA development tools to start somewhere around level 3, with some approaching level 4. Following code generation, required hand coding will be applied to the output. In the compile step that follows, all of the artifacts are compiled by the system, automatically. Executable modules are then created, also automatically, in the usual way.

The many three-letter abbreviations in this chapter may be confusing. The situation is not difficult at all though. What this chapter was describing was actually almost the same as the previous chapter, only from a little bit more complicated point of view. The CIM, Computation Independent Model, is what should be accomplished in a business modeling workflow. The PIM, Platform-Independent Model, is what should be actualized in the requirements workflow based on the CIM. If the software development project is not using the MDA, this part is the different one. The PIM is platform independent in order to allow use in similar business situations more easily, the model to be evolved further into different platforms more easily and the model to live longer. If MDA is not used, such needs probably do not exist and the PIM and the PSM are more or less the same model, remembering that the PSM was a platform specific model that considered the technology aspects. The PSM is accomplished in analysis and design phases of the software process based on the PIM. If the PIM does not exist, the PSM is accomplished in requirements, analysis, and design phases based on a CIM. The code generation mentioned in the previous chapter can be possible from the PSM whether it is achieved by the PIM or not. However, if the PIM does not exist, modifying the PSM into new, possible technical environments demands a lot more work.

Next, Meta-Object Facility (MOF) is introduced. MOF offers means to express the models described in this chapter.

3.2.2 Meta-Object Facility (MOF)

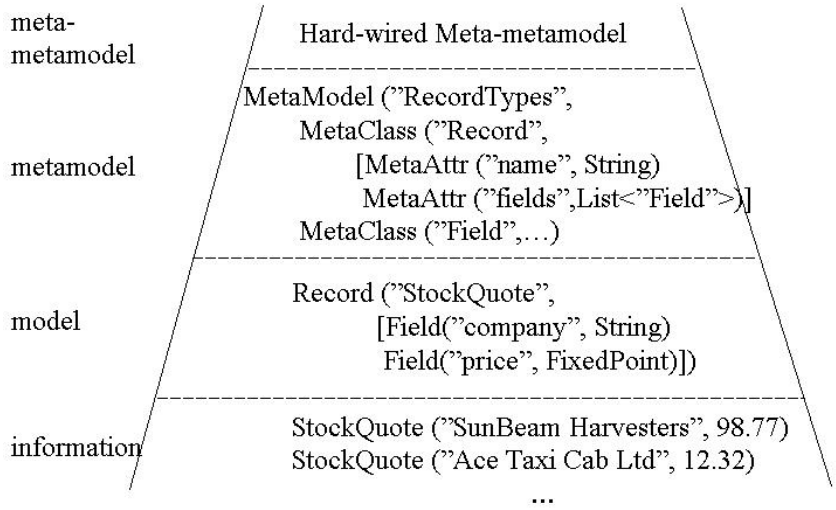
Object Management Group's (OMG 2003, p. 25) Meta-Object Facility, MOF, is intended to support a wide range of usage patterns and applications. MOF usage patterns can be understood with two distinct viewpoints: (1) From designer's modeling viewpoint, the MOF is used to define an information model for a particular domain of interest. This definition is used to drive subsequent software design and/or implementation steps for software connected with the information model.
(2) From the programmer's data viewpoint, the product of MOF is used to apply the distributed computing paradigm to manage information corresponding to a given information model.

The MOF specification defines a core MOF model that includes a relatively small set of constructs for object-oriented information modeling, which can be extended by inheritance and composition to define a richer information model that supports additional constructs. Alternatively, the MOF model can be used as a model for defining information models. In this context, the MOF Model is referred to as a meta-metamodel because it is being used to define metamodels, models of models, such as the UML. (OMG 2003, p. 26)

Extensibility is the central theme of the MOF approach to management of metadata, data of data. The aim is to support any kind of metadata, and allows new kinds to be added as required. In order to achieve this, the MOF has layered metadata architecture. Traditional four-layer metadata architecture is briefly described below. This is followed by a more detailed description of how this maps onto the MOF metadata architecture. (OMG 2003, p. 33 - 34)

The classical framework for metamodeling is based on architecture with four layers, see picture 15 below. OMG (2003, p. 34) describes these layers as follows:

- The information layer is comprised of the data that is to be described.
- The model layer is comprised of the metadata that describes data in the information layer.
- The metamodel layer is comprised of the descriptions (i.e., meta-metadata) that define the structure and semantics of metadata.
- The meta-metamodel layer is comprised of the description of the structure and semantics of meta-metadata.



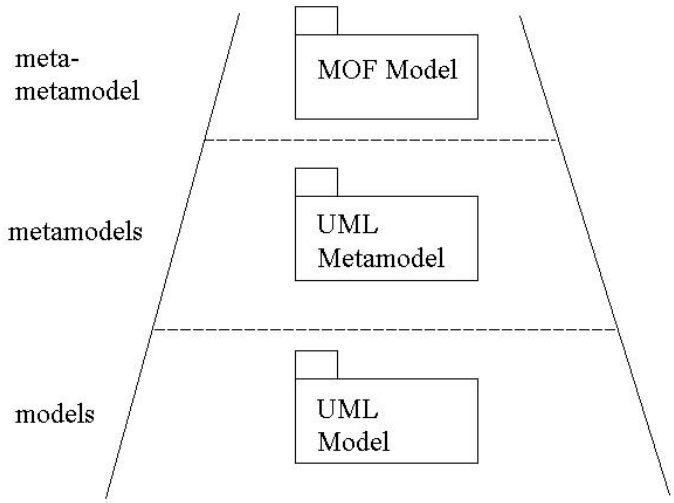
Picture 15. Four layer metadata architecture (adapted from OMG 2003, p. 34).

The example in picture 15 shows how information turns into metadata and metadata turns into meta-metadata. In the model level, field “company” refers to “SunBeam Harvesters” and “AceTaxi Cab Ltd”, the field “price” accordingly. In the metamodel level, metaclass “Record” refers to text “Record” in the model level. Accordingly, MetaAttr “name” refers to the recordname “StockQuote” in the model level. MetaAttr “fields” refers to the two fields: “company” and “price” in the model level. Since there are several fields in the model level, the attribute “fields” is of type list. Metaclass “Field” would then describe the field attributes in the model level. For example as follows: MetaClass (“Field”, [MetaAttr (“name”, String) MetaAttr (“Datatype”,…)]).

The MOF metadata architecture (picture 16, p. 40) has a few important features that distinguish it from earlier meta-modeling architectures (OMG 2003, p. 36):

- The MOF’s core meta-metamodel is object-oriented, and the metamodels are constructed equally with UML object models. Hence, the example uses UML package icons to denote MOF-based metamodels as well as UML models.
- The meta-levels in the MOF metadata architecture are not fixed. While there are typically four meta-levels, there could be more or less than this, depending on how MOF is deployed.
- A model is not necessarily limited to one meta-level. For example, in a data-warehousing context, it may be useful to think of the meta-schema “Relational table” and specific schemas that are instances of relational tables as being one conceptual model.

- The MOF Model is self-describing. In other words, the MOF Model is formally defined using its own meta-modeling constructs.



Picture 16. MOF metadata architecture (adapted from OMG 2003, p. 35).

So far in chapter three, the requirements engineering process within a software engineering process has been examined. The MDA and MOF have been introduced as means to develop platform- and technology independent models. Next chapter introduces Unified Modeling Method (UMM), which is a method that can be used to guide requirements engineering process.

3.3 Unified Modeling Method in Requirements Engineering

The Unified Modeling Methodology (UMM) is a modeling technique, which is concerned only with the inception and elaboration phases of the software development, not the implementation and transition phases (see picture 12 on page 29). The UMM is based on customizing the Rational Unified Process (RUP) methodology (Clark 2000, p. 2). The requirements engineering part of the RUP was described in chapter 3.1. Both UMM and RUP use UML as modeling technique. The UMM was developed by the United Nations (UN) / Centre for facilitation of Practices and Procedures for Administration, Commerce and Transport (CEFACT) (Clark 2000, p. 2).

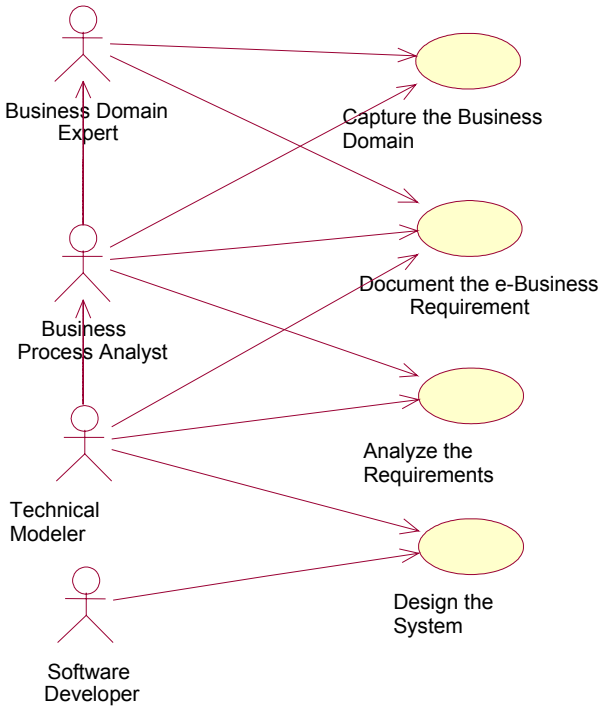
Clark (2000, p. 2 – 6) lists four workflows the UMM consists of:
1. Business Modeling aims at deriving the high level requirements needed to support the subsequent activities and making sure that all key participants have a common understanding of the business. Business Modeling delivers a

Business Use Case Model and Business Object Model (Business Entities, Business Workers). Roles participating in the workflow can be for instance business domain experts and business process analysts. Business modeling was also explained earlier in chapter 3.1.2.

2. E-Business requirements workflow takes the use cases modeled in the Business Modeling workflow, and refines the output for the business area. It concentrates on specifying requirements to a level that is good enough for the users and standards developers to agree on what the solution should provide. The deliverables of this workflow are a refined use case model, a class diagram including attributes of all the business entities and a supplementary specification of the non-functional requirements. Roles participating in the workflow can be for instance business process analysts and technical modelers. The requirements workflow was examined already in more detail in chapter 3.1.3.

3. Analysis workflow transforms the requirements into precise object oriented specification. It aims to generalize the use cases where possible to aid re-use and to build a set of common business classes. The workflow delivers a system class diagram and a set of analysis use cases including the corresponding class diagrams and interaction diagrams. Roles participating in the workflow can be for instance business process analysts and technical modelers.

4. Design workflow aims to convert the output from the analysis workflow into required definitions to enable the developing of the software solution. The workflow deliverables are message designs, object model, use case model and interface class operations. Roles participating in the workflow can be for instance technical modelers and software designers. See picture 17 for the different UMM workflows.



Picture 17. The Unified Modeling Method (adapted form Clark 2000, p. 3).

The UMM specifies activities to be performed, roles of participants, techniques to be used and deliverables to be produced. The UMM is needed to enable adopting a common approach to specification of business requirements and data so that they can be shared internally and provided externally in a consistent manner (Clark 2000, p. 2). The business and transactional processes as well as service interfaces in the DBE will be modeled using UMM (Dini et al. 2003, p. 8).

4 Open Source Software

This chapter introduces the reader with open source software (OSS) and the characteristics of OSS development. The concept of open source software is discussed first. Next, the most common OSS licenses are introduced. Then, the characteristics of OSS development are examined. Finally, after examining requirements engineering in chapter three and given the reader a general idea of OSS in this chapter, a preliminary requirements engineering recommendation for the collaboration of proprietary software company and OSS project is proposed.

4.1 What Is Open Source Software?

The source code of a program has traditionally been of special value to the software producing companies. The source codes have been available to nobody but the owning company, which is from where the term closed source software originates. According to Parviainen (2004), the point in open source software is that source code is accessible to everyone. It is also allowed to make changes to the program and compile it to one's own computer. In addition and extension to those qualities, there is Open source initiatives (OSI 2004c) definition of open source in 10 clauses (Table 1):

Table 1. Open source initiatives definition of Open source software (OSI 2004c).

1. Free Redistribution The license shall not restrict any party from selling or giving away the software as a component when distributing software, containing programs from several different sources.
2. Source Code The program must include source code, and must allow distribution in source code as well as compiled form. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a pre-processor or translator are not allowed.
3. Derived Works The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.
4. Integrity of The Author's Source Code The license may restrict source-code from being distributed in modified form *only* if the license allows the distribution of "patch files". The license must explicitly permit distribution of software built from modified source code.

5. No Discrimination Against Persons or Groups The license must not discriminate against any person or group of persons.
6. No Discrimination Against Fields of Endeavor The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.
7. Distribution of License The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
8. License Must Not Be Specific to a Product The rights attached to the program must not depend on the program's being part of a particular software distribution.
9. License Must Not Restrict Other Software The license must not place restrictions on other software that is distributed along with the licensed software.
10. License Must Be Technology-Neutral No provision of the license may be predicated on any individual technology or style of interface

Clause 1 states one of the most noted characteristics of open source software, which is that the software must be distributable freely. Noticeable is that it also allows selling of open source software as part of aggregated software. Another well-known feature of open source software is that whenever distributing programs, the source code must be somehow included, which is the message in clause 2. The modification of a program has to be easy, that is why it is emphasized that the source code must be in a clear, non-converted form. Clause 3 allows the developers of open source code to make modifications and distribute them under the same terms as the original software. This is especially important for independent peer review of code and in order to gain rapid evolvement of software. Clause 4 adjusts the definition of clause 2. It says that the source code must be available, yes, but certain open source licenses (described in more detail in Chapter 4 .2) may require that the source code be distributed as base sources plus patches. In this way, "unofficial" changes can be made available but distinguished from the base source. The open source community wants to attract as many developers as possible to gain maximum competence and speed from the process, which is why Clause 5 states that no discrimination against any person or group is allowed. Clause 6 points out that open source software is allowed to be used in any field and in any business. Clauses 7-10 express that an open source license may not be restricted by or depend on other licenses, products, software or technologies.

4.2 Open Source Software Licenses

With the current legal framework, the license under which a program is distributed defines exactly the rights that its users have over it. For instance, in most proprietary programs the license withdraws the rights of copying, modification, lending, renting and use in several machines. In fact, licenses usually specify that the proprietor of the program is the company, which publishes it and just sells restricted rights to use it. In the world of open source software, the license under which a program is distributed is also of paramount importance. Usually, according to Daffara et al. (2000), the conditions specified in licenses of open source software are the result of a compromise between several goals, like

- guaranteeing some basic freedoms of redistribution, modification and use to the users,
- ensuring some conditions imposed by the authors, for instance, citation of the author in derived works, and
- guaranteeing that derived works are also open source software.

Authors can choose to protect their software with different licenses according to the degree with which they want to fulfill these goals. In fact, authors can distribute their software with different licenses through different channels and prices (Daffara et al. 2000). Therefore, the author of a program usually chooses very carefully the license under which it will be distributed and users, especially those who redistribute or modify the software, have to carefully study its license. Fortunately, although each author could use a different license for her programs, the fact is that almost all open source software uses one of the common licenses (GPL, LGPL, BSD, MPL, etc.), sometimes with slight variations.

The characteristics a software license to qualify as open source software license have been defined by several organizations. One of them is the Debian Project, which defines the Debian Free Software Guidelines[1] (DFSG). The other is the Open Source Initiative (OSI) (covered in chapter 3.1), which is based on the DFSG (Daffara et al. 2000). The Free Software Foundation (2003b) also provides its own definition of free software. Free software refers to four kinds of central freedoms for the users of the software:

- 0: The freedom to run the program for any purpose.
- 1: The freedom to study how the program works, and adapt it to your needs.

[1] More information in http://www.debian.org/social_contract.html#guidelines.

- 2: The freedom to redistribute copies so you can help your neighbor.
- 3: The freedom to improve the program, and release your improvements to the public, so that the whole community benefits.

There is even a rule, copyleft, which protects the central freedoms. Copylefting a program means you cannot add restrictions to deny other people the central freedoms, when redistributing the program (Free Software Foundation 2001b). Therefore, it is allowed to modify the original program, share or not share the modifications with other developers and even turn the original program into proprietary use.

The difference between free software and open source software exists; however, it is not obvious. Free Software Foundation (2001c) claims that the official definition of open source software, as published by the OSI (discussed in chapter 3.1), is a little looser in some respects, and that OSI has accepted a few licenses that Free Software Foundation considers unacceptably restrictive of the users. The obvious meaning for the expression “open source software” is “You can look at the source code”. This is a much weaker criterion than free software; in addition to free software, it also includes semi-free[1] programs, and even some proprietary[2] programs. Nevertheless, Free Software Foundation admits that although it disagrees with OSI on the basic principles, the ethics and the values, the two more or less agree on the practical recommendations and work together on many projects against proprietary software, which is considered the enemy. Anyway, the two are increasingly handled together more than separately. As an example of that, lately the use of the terms Free and Open Source Software (FOSS) and Free/Libre Open Source Software (FLOSS) have grown (see e.g. Scacchi 2004, p. 1, International Institute of Infonomics 2002).

4.2.1 The GPL and LGPL Licenses

One of the most common open source licenses is the GNU General Public License. GNU GPL was motivated by an idealistic goal: spreading freedom and cooperation. Richard Stallman (in Free Software Foundation 2003a) says he wanted to encourage free software to spread, replacing proprietary software that forbids cooperation, and thus make our society better. The GPL was carefully designed to promote the production of more free software, and

[1] Semi-free program is not free, but come with permission for individuals to use, copy, distribute, and modify for non-profit purposes.

[2] Use, redistribution or modification of proprietary programs is prohibited, or requires you to ask for permission.

because of that, it explicitly forbids some actions on the software, which could lead to the integration of GPLed software in proprietary programs. The main characteristics of the GPL are the following:

- it allows binary redistribution, but only if source code availability is also guaranteed;
- it allows source redistribution (and enforces it in case of binary distribution);
- it allows modification without restrictions (if the derived work is also covered by GPL);
- complete integration with other software is only possible if that other software is also covered by GPL. (Free Software Foundation 1991)

The GPL allows the selling of copies of the program for money, the right to sell copies is part of the definition of free software. The GPL also allows charging a fee for downloading the program from an internet-site. The fee may be whatever the distributor wishes, however, since an “equivalent access” to the source code must be provided, the fee to download source may not be greater than the fee to download the binary. Modifying a GPL-licensed program into a new program and then distributing it commercially is allowed too, but only under the terms of GPL. (Free Software Foundation 2001a) Thus, if GPL-licensed software is distributed for a fee, the GPL gives the receivers the freedom to release it to the public, with or without a fee. Therefore, if a business included only selling GPL-licensed software, it would seem to be a quite impossible job to make it profitable. However, using GPL-licensed software for commercial purposes without ever releasing the possible modifications made, is perfectly possible (Free Software Foundation 2001a). On the other hand, dual licensing, licensing the same software under both open source software license and proprietary software license, could be a business possibility for the original copyright owner of GPL software. It is allowed only for the copyright owner to license the software under many different licenses including also GPL (Free Software Foundation 2001a).

There exists a less restrictive version of GPL license, which is Lesser or Library General Public License (LGPL). LGPL license is suited for needs to include software into proprietary software, which the GPL license strictly prohibits. (Free Software Foundation 1999): “This General Public License does not permit incorporating your program into proprietary programs.” “If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this (GPL) License.”

4.2.2 The BSD and MIT Licenses

Berkeley Software Distribution (BSD) license is simpler than the GPL. It is shorter and has actually only few restrictions. The BSD permits redistribution and use in source and binary forms, with or without modification, if the following conditions are met:

- redistributions of source code and redistributions in binary form must retain or reproduce the above copyright notice, this list of conditions and the following disclaimer in some material provided with the distribution and
- the project owners and its contributors must be asked to give written permissions if derivative software is to be marketed with their names (OSI 2004a).

The original BSD license was incompatible with the GPL, since it imposed a restriction on advertising the software. The clause originally read “All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by the University of California, Berkeley and its contributors (Hoskins 1999).” This clause provided a restriction on the recipients exercise of the rights granted in the GPL license, thus it was GPL-incompatible (Free Software Foundation 2001a). The advertising clause was deleted by William Hoskins (1999), the Director of the Office of Technology Licensing of the University of California in 1999, so the two licenses are now compatible.

The BSD is a good example of a permissive license, which imposes almost no conditions on what a user can do with the software, including charging clients for binary distributions, with no obligation to include source code. The authors only want their work to be recognized. This restriction ensures a certain amount of “free marketing", in the sense that it does not cost money, but it is important to notice that this kind of license does not include any restriction oriented towards guaranteeing that derived works remain open source.

The MIT license does not forbid using the names of the contributors and the copyright owner without their written permission in marketing the software, otherwise it is similar to the BSD. The license gives permission to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the software, and to permit persons to whom the software is furnished to do so. The one condition is that the copyright notice and the permission notice shall be included in all copies or substantial portions of the software. (OSI 2004b)

4.3 Working with Open Source Software Community

4.3.1 General Characteristics of Open Source Development

Open source software engineering has several characteristics some of which are rather different from proprietary software engineering. Open source projects operate in the Internet and communicate by using posting lists and development tools (Hansen et al. 2002, p. 465-466). The projects are virtual in the sense that developers might never meet each other face-to-face. Thus, the software is developed decentralized in multi-site co-operation, and it has to be modularized to make the development possible (Lerner and Tirole 2001, p. 823). Concurrent Versions System (CVS) or similar systems are used for managing the changes and the reasons for changes in the code. With a tool like the CVS, the changes are as a rule step-by-step reversible. (Hansen et al. 2002, p. 465-466)

Although open source projects seem to be unmanageable and dispersed, they are based on a hierarchical structure. Fink (2003, p. 55) states that a maintainer, who often is an individual even in large projects, runs most projects. The maintainer accepts or rejects the submissions and patches the developers have suggested. The maintainer is not just anybody willing to do the job. He or she is the individual either who initiated the project or who was assigned for the job by a previous maintainer. A new maintainer is always selected from a list of volunteer individuals, who have made significant contributions to the project. By contributing high-quality code often, an individual can build respect for himself/herself in the open source community and rise in the hierarchy. The maintainer will more readily accept contributions from someone with a history of good, solid code. Hansen et al. (2002, p. 466) say contributions to modules and other communication are done through a mailing list, which is available usually for everyone for reading. Contributions from unknowns will be carefully examined and if poor code is found, the criticism will be vocal (Fink 2003, p. 55). Fink (2003, p. 139) calls this “built-in talent management”, meaning, if you do not make significant contributions to the open source community, the community does not want you.

According to Raymond (1998) releasing software packages early and often is a critical part of the open source development model. Although early releases contain many bugs the developers normally do not want the users to see, the open source community has succeeded in making this development style work. Namely, in the open source community the developer is the user (Fink 2003, p. 140). Therefore, early releases can be made for the qualified and numerous users who will soon provide patches and other code suggestions for the maintainer to be evaluated and released in a new package. As Raymond (1998, p. 1) puts it “given enough eyeballs, all bugs are shallow”. Fink (2003, p. 56)

states that early detected and fixed defects are one of the reasons for the higher quality of open source software compared with traditional products. Moreover, higher quality is a result of releasing the software to production when the maintainer thinks it is ready for it, not because of timeline driven external factors that traditionally have had their impact on the release decision, but which unfortunately have very little to do with how ready the product is. On the other hand, some mature open source projects have time-based releases, which does not mean however, that the maintainer would have to release the software even if it were not be ready. Additionally, there is a risk that no community forms around software that is licensed as open source (Fink 2003, p. 198). The community may also be very small, just a few people. In that case, the amount of “eyeballs” is significantly reduced and detecting errors is as difficult and time-consuming as it is anywhere. Therefore, it cannot be said that the quality of open source software is always better than proprietary software.

Raymond (1998, p. 1) calls the open source development style described above the Bazaar. The traditional style of development, which most proprietary companies use, where teams are kept small, design and functionality are well agreed upon before development begins and release cycles are few, Raymond calls Cathedral development. Actually, as stated earlier (chapter 3.1.1) there are different software development styles in the proprietary world, like sequential and iterative processes. What Raymond was describing, pictured more the sequential process. The iterative development style is approaching the Bazaar style, as it has several development cycles. The duration of a cycle in an iterative process is much longer though; open source cycles are sometimes even hourly (Fink 2003, p. 138), whereas the proprietary iterative development may reach weekly cycles at its best. On the other hand, if an open source project is not challenging enough to attract a sufficient amount of developers, the rapidity of the development cycles will noticeably fall and the two worlds are not that far apart from each other.

4.3.2 Open Source Software Development Styles

The Bazaar style of software development was demonstrated by Linus Torvalds when the Linux operating system was developed in the early 90’s (Raymond 1998, p. 2). Fink (2003, p. 53) adds that the Bazaar style of development is the basis of all open source development and it has been adapted to different development styles: company, foundation, committee, individual and Linux kernel. Next, a short description of the different development styles is given.

In a company style open source development, a corporation like for example HP, IBM, and SUN, takes the lead for a project. The corporation assigns paid

developers to the project to host the development. Often the maintainer of the project in a case like this comes from the corporation and is an individual. (Fink 2003, p. 53) On the other hand, Fink (2003, p. 55) adds that the maintainer may also be a group of individuals in that company. A foundation is a non-profit organization, which is created to fund a large project. Funds are donated by companies and individuals who are interested in making sure the project is successful. Sometimes a foundation is established for funding the project but the lead of the project is taken by a company. In that case, the company might not have had the financial resources to fund the project on its own. (Fink 2003, p. 53 - 54) A committee is set up for a large project too. Committee is established to provide a forum for decision-making, thus, in a committee style open source project the maintainer is not the single decision maker. The GNU C/C++ Compilers are managed this way. Individuals run small projects that typically involve less than five developers. There are thousands of these project and many of them are hosted on a site called SourceForge. (Fink 2003, p. 54) SourceForge.net provides free services to open source developers, over 822 000 developers have registered to the site. SourceForge.net is the largest open source software development site in the world, it is hosting over 78 000 projects. (SourceForge™.net 2004) Finally, Fink (2003, p.54) says that Linux kernel development style is used by Linus Torvalds for the largest open source development project. It is based on a hierarchy of trust within the development community, which has evolved for years.

4.3.3 Open Source Software Requirements Engineering

The elements of a software engineering process can be named in many different ways but the contents of those element do not vary that much. In the third chapter the software engineering process elements were called requirements, analysis, design, implementation and test (see picture 12 in page 29). Dibona et al. (1999, p. 91) entitle the elements as marketing requirements, system-level design, detailed design, implementation, integration and field-testing. The latter list is a bit more accurate since it has separated integration, compiling and linking separate modules into one package, into its own element. Otherwise, the names of the elements are interchangeable. Dibona et al. (1999, p. 96) state that roughly said, there are two kinds of open source projects: those, which include every single one of the above elements, and those that are not so well organized. Next, the elements of the software engineering process and requirements engineering process are taken a closer look from the open source development point of view.

The starting point in requirements eliciting in an open source project is that in an open source project the developer is also the user of the system (Fink 2003, p. 140). DiBona et al. (1999, p. 96) agree as they say that open source people develop software they need or wish they had. Discussion about the requirements is usually done with mailing lists or in newsgroups. Scacchi (2001, p. 8) agrees as he states that apparently open software requirements are articulated in a number of ways and that they are ultimately expressed, represented, or depicted on the Web. On closer examination, requirements for open source software can appear or be implied within an email message or within a discussion thread that is captured and/or posted on a project's Web discussion board for open review, elaboration, refutation, or refinement. These requirements are simply asserted without reference to other documents, sources, or standards; they are requirements because some developers wanted these capabilities. Asserted system capabilities are post-hoc requirements characterizing a functional capability that has already been implemented. The concerned developers justify their requirements through their provision of the required coding effort to make these capabilities operational. On the other hand, requirements are not anymore received only from individual developers, open source projects have recently begun to accept requirements from companies that have invested resources in the project-

Agreeing with what is implemented and what is not is not always easy. If true consensus is not reached often enough, developers might split, start developing and releasing their own versions. (DiBona et al. 1999, p. 96) Fielding (1999) says it is the senior members or core developers in the community that vote or agree through discussion to include the already implemented capability into the system's distribution. So, there seem to be two different times, when a decision about including a requirement into a software is made: DiBona et al. say the decision is made before the implementation and Fielding adds it is made after the implementation. Anyway, when a requirement has been implemented and included in the software package, Fink (2003, p. 140) says that since the developer is the user, the requirements tend to meet the user's needs. However, nowadays there are consumers of open source that are not developers, thus, the requirements do not necessarily meet their needs after all. Alternatively, the consumers of open source might be satisfied with the software but have new ideas they would like to be implemented by the community, since they themselves might not have the skills to do that. That is one of the reasons for this study to be made.

Scacchi (2001, p. 8 - 9) states that there may be a requirements history available, within the email or discussion board archive, to document who required what, where, when, why, and how. However, once asserted, there is generally no further effort apparent to document, formalize, or substantiate such

a capability as a system requirement. Asserted capabilities then become invisible or transparent, taken-for-granted requirements that can be labeled or treated as obvious to those familiar with the system's development. Scacchi (2001, p. 10) has found another different kind of open source software requirement; a requirements “vision” document that conveys a non-functional requirement for “community software development”. The community software development, and by extension, community development, may be recognized as being important to the development and success of the system. Moreover, in the Internet/Web infrastructure community there is evidence for elicitation of volunteers to come forward to participate in community software development.

Requirements analysis for an unfunded, small open source project usually does not exist or it evolves over time. There might be an implicit design in the head of a developer by version two or three of the system, even if it is not written down anywhere. (DiBona et al. 1999, p. 97) Spinellis & Szyperski (2004, p. 4) offer a completely different point of view. They say that open source developers, while working in large, organized open source projects, learn best-of-breed software engineering practices and that they even diffuse those practices to other, possibly proprietary projects, which they undertake for pay. Scacchi (2001, p. 16) notes that requirements for open software system are, in practice, analyzed via the reading of technical accounts as narratives, together with making sense of how such readings are reconciled with one’s prior knowledge. However, the functional and non-functional requirements in the technical accounts are not self-contained. Instead, each requires the reader (e.g., a developer within the community) to closely or casually read what is described, make sense of it, consult other materials or one’s expertise, and trust that the description’s author(s) are reliable and accountable in some manner for the open software requirements that have been described (Goguen 1996, Pavlicek 2000). Analyzing open software requirements entails little if any automated analysis, formal reasoning, or visual animation of software requirements specifications (Nuseibeh & Easterbrook 2000). Yet, Scacchi (2001, p. 16) observes, the participants in these communities are able to understand what the functional and non-functional requirements are in ways that are sufficient to lead to the ongoing development. It is becoming increasingly apparent that open software requirements can emerge from the experiences of community participants through their email and discussion forums. These communication messages in turn give rise to the development of narrative descriptions that more succinctly specify and condense into a web of discourse about the functional and non-functional requirements of an open software system. This discourse is rendered in descriptions that can be found in email and discussion forum archives, on community Web sites, and in other informal software descriptions that are posted, hyperlinked, or passively referenced through the assumed common knowledge that community participants expect their cohorts to possess.

DiBona et al. (1999, p. 97 - 98) argue that in the open source community requirements design is as unwanted job as requirements analysis was. At least that is the case in small, unfunded projects. If detailed design is written, it is written after the implementation and it takes the form of manpages or similar. Scacchi (2001, p. 17) states that apparently the requirements for open source software are co-mingled with design, implementation, and testing descriptions and software artifacts, as well as with user manuals and usage artifacts. Similarly, the requirements are spread across different kinds of electronic documents including Web pages, sites, hypertext links, source code directories, threaded email transcripts, and more. Anyway, the requirements are described, asserted, and implied informally. Yet, in threaded emails and discussion boards, it is possible to observe that community participants are able to comprehend and condense wide-ranging software requirements into succinct descriptions using lean media (Yamaguchi et al. 2000) that pushes the context for their creation into the background.

DiBona et al. (1999, p. 97 - 98) argue that implementation is what programmers love the most and what motivates them the most. An unfunded open source project can have rigor and consistency but usually they lack formal peer review, unit tests and regression. Integration involves writing some manpages, making sure the project builds on systems the developer has access to, cleaning up the extra hair that has crept in during the implementation phase, putting the project available somewhere and posting a note to a mailing list so that interested people can fin it. However, integration phase does not include system-level testing. Testing in unfunded open source projects is done by numerous developers / users in the field. Testing is carried out both by executing the packaged executables, but also by peer-review, programmers reading the code trying to find bugs. How are requirements validated then or are they? Scacchi (2001, p. 17) finds that requirements validation is an implicit by-product, rather than an explicit goal, of how open software requirements are constituted, described, discussed, cross-referenced, and hyperlinked to other informal descriptions of system and its implementations.

4.4 Requirements Engineering Collaboration

The two environments, proprietary software development and open source software development, seem to be very different regarding requirements engineering. When composing a requirements engineering model for the collaboration of a proprietary company and an open source project, which in this case is the DBE infrastructure, there are certain issues to be into consideration.

First, the DBE is a large project, therefore, the characteristics of small open source projects do not count in this case. Secondly, the collaboration is approached from the ongoing open source project point of view, since the DBE is one. Thus, for a SME, constructing an open source project of its own is an option not taken into account here. Thirdly, the assumption made here is that the proprietary software producing SME is utilizing open source software and has important needs it wants the open source project to implement in the software. The collaboration proposal presented next, is handled by the characteristics introduced earlier in chapter 4.3.

4.4.1 Communication Infrastructure

Most of the requirements engineering work in the proprietary software world is carried out face-to-face. If that is not possible, at least regular meetings concerning the subject are arranged. In the open source community, the participating developers probably never meet each other. The requirements engineering communication is performed in several Web-based forums, according to Scacchi (2001, p. 18) they are: (a) messages placed in a Web-based discussion forums; (b) email list servers; (c) network news groups; or less frequently in (d) Internet-based chat. Messages written and read through these systems, together with references or links to other messages or software webs, then provide some sense of context for how to understand messages, or where and how to act on them.

When considering co-operation with the open source community, the obvious solution for the communication infrastructure would be Web-based forums. After all, they are a very convenient way of communicating regardless of time and location. Moreover, they are the best way to reach the numerous developers of open source community. On the other hand, if the communication seems disconnected the number of forums used could be cut down in order to reach a more coherent community.

4.4.2 Decision Making in Open Source Community

The customer and his needs are in the leading role in the proprietary requirements engineering. The customer is the user of the software to be composed, therefore he is the expert who decides what is required, what is implemented and what not, and in which order. In the open source community, for a long time it has been the developer who is the user of the software. The best and most productive developers may rise in the hierarchy to become maintainers of open source projects. The maintainers have all the power over the projects. In small open source projects, there may be just one maintainer, in

larger projects there are lower level maintainers responsible for their own areas in addition to the maintainer or a maintainer group at the highest level. Lately, it has been noted that the use of open source is increasing also elsewhere besides among open source developers. How the community considers that, or does it, is not very clearly definable yet.

When collaboration between a proprietary company and open source community is planned, it would be wise to notice that the existing, long developed decision making structures of open source community may be very difficult, if not impossible to change. Moreover, the maintainers have earned their position with quality work, thus, they can be assumed capable for the job. Therefore, if the company were participating in an existing open source project, it would be a good idea to follow the rules of the community where the maintainer makes the decisions. Thus, although the company is the user of the open source software, it automatically has very little, if any power over the project, at least in the beginning. Noticeable is that the maintainer also makes the decision about when the project is ready to be released for production. Timelines cannot be given for an open source project to be followed.

4.4.3 Power Relations

In the proprietary software world, power is attached to the position and position is gained as a sum of many things: education and work experience mainly. On one hand, quality of work may be an asset in gaining a position, on the other hand, bad quality not always is an obstacle when trying to advance in one's career. In the open source community, power is gained by contributing quality code and doing it often. Position an open source developer may have outside the community, has no affect on power in the community. The number of accepted contributions is what counts and causes a rise in the decision-making hierarchy. Bad quality is not tolerated.

When a company is considering co-operation with the open source community, it probably has some ideas it needs or wants the community to develop into a software product or a software feature, and with low (or no) costs, of course. If the community would take orders from the company, that would be simple. However, that is not the case. In addition to how power is gained in the community, the company should bear in mind the basic idea of open source: if you take something from the community, it would be advisable to give something back too. Therefore, if the company truly wants the open source community to develop the very features of the software it has in mind, it would be a good idea to participate in the work. Hiring a talented software engineer in the company to develop the software with the open source community could do

that. In time, the developer may gain more and more power over the project and even before that, if the developer is talented enough, the needs of the company may be accepted as part of the software by the maintainer.

4.4.4 Business Modeling

Business Modeling is the first step when developing proprietary software. The goal is to specify the business entities the system is going to support. Business model is described in Unified Modeling Language (UML) as use-case models and use cases. The business model is then used as a starting point when deriving the first requirements. The business model is further developed as sharpening the requirements.

Business Modeling is simply not done in the open source community. At least, it was not once mentioned in the references studied for this research. Of course, it may be that suitable references were not found. Anyway, describing and documenting software features in advance in not strength of the open source community. Therefore, when collaborating with open source projects, it would be advisable for a proprietary company to model the business itself. The documents produced would then be saved in the open source project Web site or another Web-based communication forum, where the documents can be reached and further developed by open source developers.

4.4.5 Requirements, Analysis and Design

Requirements workflow is carefully performed and documented in the proprietary software world. There, the use-cases depicted in the Business Modeling workflow are further developed. Analysis and Design phases develop the requirements even further to ease the beginning of implementation. In the open source community too, requirements are discussed, analysis is performed and designs of the system are drawn. It is only that the requirements are discussed probably not modeled, analyzing happens, but likely only in the head of an individual developer, and designs are written in the form of using instructions after the implementation, they are not modeled before the implementation. To put it shortly, in open software development projects, requirements engineering efforts are informal by-products of other activities.

So, when a company wants to co-operate with an open-source project and the requirements engineering efforts to be performed in a more formal manner, the job has to be done by the company itself and the documents have to be saved

in the project Web site. There, the other developers may give their expertise to the designs if they want to and use the documents to ease the implementation.

Noticeable is that even if the company collaborating with an open source project contributes a lot of documents to the project, it is the implementation and the quality code that impresses the community, builds trust in the developer and eventually, may result in a powerful position in the community hierarchy. Therefore, it would be advisable for a company to participate actively in the implementation phase too.

5 Case: DBE Infrastructure and Pilot Applications

This chapter introduces the reader to the Digital business Ecosystem (DBE) – project. First, the project objectives are stated in general. Next, the twelve subprojects are shortly introduced. Finally, the DBE infrastructure and its pilot applications are introduced.

5.1 Digital Business Ecosystem - project

The Digital Business Ecosystem - project aims to provide European SMEs with innovative software development and recognised advantage and to achieve greater ICT adoption in general. The objective of the DBE project is to develop an open source distributed environment that can support the evolution and composition of (not necessarily open source) software services, component and applications. Thus, one of the project outputs will be a component-based, open source software infrastructure.

The DBE will adopt a multi-disciplinary approach (Dini & Nicolai 2003, p. 2). In fact, according to Anon (2003, p. 5) the DBE vision rests on two fundamental ideas: those of self-organization and biological evolution. The concept of self-organization implies intelligent behavior and the ability to learn on a short time scale, whereas evolution implies an ability of the system to optimize itself through differentiation and selection of its components on a long time scale. The term DBE can be understood and applied at different levels. It refers to:

- The open-source, distributed environment (infrastructure and middleware) within which components and services interact and evolve.
- The environment plus the core components and services that allow discovery and evolution, the life-support layer of the DBE
- The environment with all the core components and services, plus the applications that are actually used by the businesses, both co-evolving.
- Finally, in its fullest sense, not only the applications and software services used by the businesses but, through coupling, the developing business models as they are influenced by the evolving software and the interactions with other businesses.

The DBE results will represent a paradigm shift in software access, distribution and integration that will change the way SMEs and European software providers will use and distribute their products and service, bootstrapping a European Digital Business Ecosystem. The four requirements fundamental to the DBE are:

- The ability to generate software structures that meet the requirements of SMEs.
- The ability to evolve and adapt software components as these requirements change over time.
- The ability to generate evolving software structures that go beyond the minimum requirements of SMEs, and which influence their business models in a positive way.
- The ability to adapt its structure to local SME ecosystems. (Anon 2003, p. 5 - 6)

The DBE project is an integration of three different domains: Business, Computing and Science (Anon 2003, p. 27). Business domain deals with for example business modelling, requirements, requirements gathering, and regional catalysts. Computing domain defines and develops a knowledge base, Business Modelling Language (BML), Service Description Language (SDL) and dynamic services among other things. (Dini & Nicolai 2003, p. 8) Science domain will produce models that regulate self-organization, resiliency based on memory, adaptability to changing environment and measures of compliance of the DBE service components with the SME user requirements (Anon 2003, p. 109).

5.2 DBE - subprojects

The Digital Business Ecosystem (DBE) project divides into twelve sub-projects (SPs). In SP1, the aim is to understand how transaction and information channels can improve the efficiency of SME markets mediated by adaptive and modular software services, and to translate these insights into algorithms that can be implemented in the form of an optimizer in the DBE recommender system. Building knowledge representations of business entities, processes, and relationships will be the starting point of SP2, DBE Language. Starting with the analysis of SME needs and requirements, ontology and knowledge bases are designed and a semantic hierarchy is constructed that will enable the expression of business declarative statements through a Business Modeling Language (BML). BML will be compiled into a Service Description Language (SDL) for describing the services that, when formed into suitable chains, can enact the required functions. (Anon 2003, p. 30 - 31) The fundamental issue that SP3, Intelligent Service Composition addresses is that, while in principle the automated representation of user requirements into a descriptive low-level language is not difficult to envision, there remains a big gulf between setting the requirements and satisfying them with some constructed algorithm. As a first

step toward the construction of such a solution the Intelligent Composer will learn over time through user profiling data which combinations of services are more likely to satisfy a particular set of requirements. The more probable usage patterns are used to refine the knowledge base, providing a feedback loop that will make the SDL specifications increasingly precise over time. (Anon 2003, p. 31) The output of the Intelligent Composer is fed to a part of the DBE that implements evolutionary optimization algorithms in the form of a fitness landscape, the subject of SP4, Evolutionary Dynamics of the DBE. This is due to that memory-based self-organization promises to be an effective approach to search the large and growing combination space of service chains, but it is not very discerning. In the initial implementation, the fitness of populations of chains of services are simply measured as a function of their "distance" from the requirements obtained from the BML compiler. This will set the stage for a more ambitious and longer-term research challenge. (Anon 2003, p. 31) SP5 is concerned with the definition of the architecture that can support these distributed and interacting parts of the infrastructure and of the software components that inhabit it. This sub-project will also explore different peer-to-peer network topologies for greater resilience, distributed storage to support the distributed intelligence and core functions and avoid a single point of failure, and security. Connected to these aspects is the Regulatory Framework for the DBE (SP10). The DBE infrastructure will be implemented in its various phases in SP6, Bootstrap and Production, and it will be populated with software services, SME users and SME providers in SP7, Population. SP8, Training, and SP9, Regional Catalysts, will look after the transfer and adoption in regional innovation clusters, and SP11, System Viability, will connect the result of this work to standards bodies and to the ERA policies and vision. (Anon 2003, p. 31 - 32)

Sub-project 9 (SP9), is performed in Tampere region. The SP9 aims at developing a model of Regional Catalysts in order to be an effective facilitator of SMEs in their integration inside the Ecosystem. The SP9 consists of two work packages (WP): WP27 – SME recruitment and WP31 – Regional Catalysts, which is the one involving Tampere region. A regional catalyst is a regional organization with deep knowledge of the SMEs and their local environment, able to involve SMEs in integration business and technical processes and enabling the SMEs to compete or collaborate on world markets (Anon 2003, p. 60).

The WP31 identifies the requirements for regional catalysts in order to support the fundamental behavior of DBE, such as self-organization, emergence and continual renewal in domain and location specific ecosystems. In addition, the work package provides networking opportunities to companies located in three areas: Tampere (Finland), Midlands (the UK) and Navarra (Spain). The Specific

Objective of this Work Package in the first 18 project months is to launch the operation of regional catalysts and support the development of BML. The WP31 consists of four tasks of which the task B29 – Functional Analysis and Specification is the first one and the one related to this thesis. The aim of the task B29 is to create a conceptual and functional description of regional catalyst and contribute to the definition of the SME requirements for DBE system. Tampere Technology Centre will perform the work in collaboration with Tampere University of Technology, as well as SMEs and other potential regional catalysts located in Tampere region. Tampere Technology Centre, as a lead partner, is directly responsible for the activities performed in Tampere Region. Other partners will be responsible for the activities in other two regions. (Anon 2003, p. 176) This thesis examines the field of SME requirements for the DBE infrastructure as part of the task B29 in the DBE project.

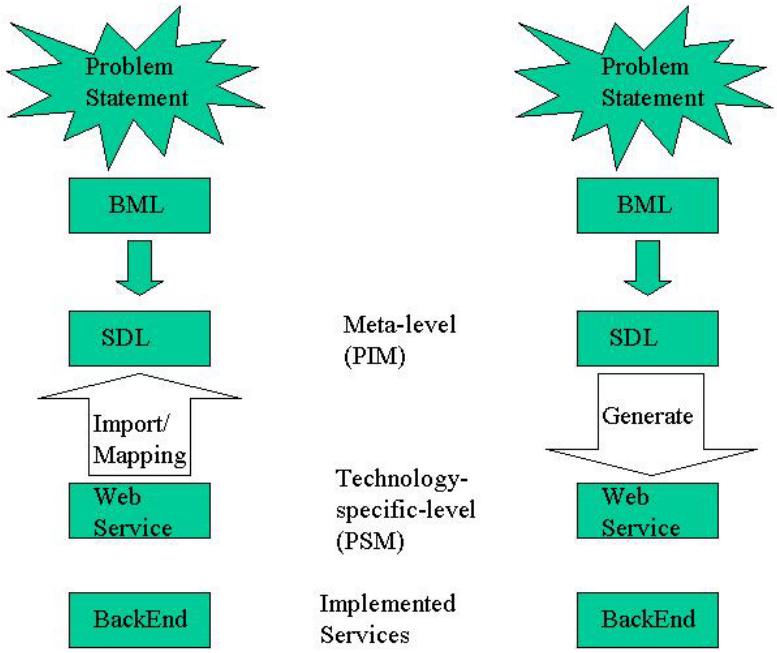
5.3 DBE infrastructure

The DBE infrastructure will be a scalable, self-managing open-source distributed platform that supports the deployment, retrieval and composition of software services. The platform will be built on a peer-to-peer distributed architecture, thus, there will be no need for centralized storage, management and control of services and data. The DBE will be self-managing with the exception of access control that will be provided by SME catalysts acting as gatekeepers and certification authorities for the platform. (Anon 2003, p. 8)

The DBE architecture is the top layer of a stack of middleware technologies. While the DBE runs over Internet Protocol (IP) and the Internet and therefore has a reasonably clear lower boundary, on the top side there is no clear boundary where the DBE ends—for instance at the user interface. This is because the dynamic behavior of the software components is coupled to the business dynamics. New business opportunities lead to new combinations of software services and, conversely, the discovery of new software service combinations will lead to new business alliances or new markets. In order to enable the dynamic aspects that allow discovery, coupling and mapping, the middleware will be leveraged applying meta-modeling infrastructure and description languages at the business, the component and the service levels. (Anon 2003, p. 27) The connection between meta-modeling infrastructure (described in chapter 3.2) and description languages is explained next.

The SME needs and requirements are analyzed and expressed through a Business Modeling Language (BML) in business declarative statements. The BML is at the business level of the DBE where many different questions, like parameters that define the enterprise and the services that implement the real-

world services and contract and agreement characteristics, have to be handled (Dini et al. 2003, p. 9). The BML is yet to be developed, but certain languages and methodologies can be listed as possible BML templates. These include ebXML[1], OWL[2] and UMM (already introduced in chapter 3.3). BML will be compiled into a Service Description Language (SDL) for describing the services that, when formed into suitable chains, can enact the SME required functions. SDL, like BML, has several template candidates, such as WSDL[3], WS-Security[4], WSCL[5] that are not exclusive. Which technologies of the above (or others) and to which extent the DBE shall use, has not been decided yet. See the connection between meta-models and description languages in picture 18.



Picture 18. Connection between meta-modeling infrastructure and description languages (adapted from Dini et al. 2003, p. 13).

The SDL description is a meta-level model of business requirements to be implemented and it is technology independent. Thus, it refers to the Platform Independent Model (PIM) (Dini et al. 2003, p. 13), introduced in chapter 3.2. From the SDL description a Platform Specific Model (PSM) is then generated, as stated earlier in chapter 3.2 in page x. The PSM can also be mapped

[1] Electronic Business using eXtensible Markup Language. More information in http://www.ebxml.org.
[2] Web Ontology Language. More information in http://www.w3.org/TR/owl-features/.
[3] Web Services Description Language. More information in http://www.w3.org/TR/wsdl.
[4] Web Services Security. More information in http://www-106.ibm.com/developerworks/webservices/library/ws-secure/.
[5] Web Services Conversation Language. More information in http://www.w3.org/TR/wscl10.

backwards into a SDL description. (Dini et al. 2003, p. 13) Software code is generated form the PSM and modified by hand.

6 Case Studies: Requirements of the Companies

In this chapter, seven cases are researched by using case study method, which was already introduced in the chapter 1.4 (p. 8). In this study, empirical examples are software-producing SMEs from Tampere region that are potential providers of software services to the DBE. First, the practical matters in conducting the research are explained. Next, requirements engineering practices in the case companies are presented. After that, the co-operation with the open source community is explained. Last, requirements engineering practices for the DBE are recommended based on the proposal made in chapter 4.4 and the empirical results.

6.1 Case Study in Practice

6.1.1 Conducting the Research

The seven case companies that were researched for this study, were chosen mainly because of three reasons. First, software-producing companies were chosen because they are the ones that are to be involved with the DBE project in the first phase producing the first services. Secondly, the companies were either operating in a very interesting field to the DBE project or had indicated their interest in participating the DBE project, in many cases both. Thirdly, it was known ahead that the company had some or a lot of knowledge about open source software development. This way, both software requirements engineering point of view and open source software development point of view could be taken into account. In addition, all of the companies were located in Tampere, which is one of the three pilot regions in the DBE project. In this study the companies are referred as company A, B, C, D, E, F and G to ensure their anonymity.

In this study, the empirical data was collected in interviews and from the company websites. Interviews were held during the first half of May 2004. One person from every company was interviewed. The person was selected by the company itself with some recommendations from the interviewer: the most suitable background for an interviewee would be a position of a project manager or equivalent, and it would be desirable if the interviewee had some experience or knowledge of requirements engineering and open source software. In all of the companies, interviews were semi-structured. Thus, the questions for the interviews were done beforehand but also additional questions were possible to make and indeed were often made. Most of the questions were so called open questions, to which interviewees could freely give their answer. In some

questions, interviewees were given examples about possible answers but the answer was not limited to those examples. Sometimes the interviewer helped the interviewees to understand the questions by picturing them the imaginary question situation. That was especially the case when the interviewees were asked about what is acceptable in open source co-operation and what is not. The interviewer also asked questions if she could not understand what the interviewees meant with their answers. If the interviewee did not know the answer to a question, the interviewee made inquiries in the company and sent the answer to the interviewer afterwards by email. If the interviewees requested, questions were sent to them beforehand, but majority of the interviewees did not demand this. Interview results were weakly validated during the interview. At the end of May, companies received a summary of the interview results for verification and possible additional comments.

Interviews were done to reach two aims: (1) to understand requirements engineering practices in the companies and (2) to find out information of the knowledge and acceptance of the open source software development characteristics. First part of the interview concerned the requirements engineering practices in the companies and experiences and expectations of co-operation with the open source community. In the second part, a list of arguments of open source development characteristics was approached twice, from different angles. First, the interviewees were asked to state whether they agree the argument to be a characteristic of open source development or not. Then, they were asked to state whether they think the argument to be acceptable or not, if their company was co-operating with an open source project. Interviews took about 1.5 hours. All of them were done face-to-face and documented in writing. Questions used in the interviews are in the Appendix 1.

6.1.2 Presentations of the Companies

The companies were all software producers from Tampere. The number of employees in the companies varied from two to eighty people, being most often around ten to fifteen people. All companies were growing or were planning and carrying out actions to grow. Following are the individual presentations of the companies.

Company A provides electronic services for order, supply, and invoicing routines, product data sharing and company's internal network management. It makes the customer processes more efficient by electrifying the connections and thereby minimizing the errors. The company is growing rapidly and has multiple private owners. There is a two-layer management in the company. Interviewee A works as development manager with technologies, product

development and the coordination of implementation as his responsibility. His work concerns customer requirements and he has good knowledge of open source software.

Company B creates and adds customer value by increasing the productivity of knowledge intensive work and the rapidity of adopting knowledge in organizations. It is a start-up-company with multiple private owners. The company has one management level. Interviewee B brings the market signals in the product development. Moreover, he works with the customer as a consultant. His job includes scanning the customer needs on a general level and it is not related to open source software.

Company C provides services that ease the customers to manage their processes, especially concerning schedule designing and cost controlling. The company has recently begun to grow rapidly, consequently, the number of employees has increased as well. The company has multiple private owners and one management level and it is profitable. Interviewee C works as manager for the employees that actually implement company services. Technical architecture of the company services includes in his responsibilities too, so does sales, as he has good connections with the field. Requirements engineering and open source software belong essentially in his job description.

Company D provides customers services that require special knowledge, which the customers do not have or it is not profitable to have, the company also performs some consulting. The company is owned and managed by one person. Interviewee D does all kinds of tasks from implementing to consulting. His work involves requirements engineering a lot and he is quite familiar with open source software as well.

Company E provides tools and services for competence management, the three areas it is involved with are: education and consulting, content production, and open source software technology services. The company has multiple private owners, one management level and it is profitable. Interviewee E is the managing director of the company. Requirements engineering includes in his responsibilities and he has a profound understanding of the open source software community.

Company F provides services and tools for events. The services and tools accelerate the customer work and increase the quality by diminishing errors. The company is owned by multiple private owners, it has one management level and it is profitable. Interviewee F works as marketing manager. Sales, requirements gathering and management, and supervision of the

implementation are included in the job description. Open source software is not very familiar to interviewee F.

Company G provides consulting, for example, for building a business intelligence strategy or an information management strategy. In addition, the company is experienced in manufacturing large-scale software systems based on advanced technological architectures. The company is growing rapidly and has multiple private owners. There is a two-layer management in the company. Interviewee G works as project manager with development of methodologies as one of his responsibilities. He is experienced in requirements engineering and he has become familiar with open source projects, as finding suitable open source components is another responsibility of his.

6.1.3 Classification of the Companies

The DBE project models and classifies small and medium sized enterprise (SME) types in order to relate business processes and needs to identifiable types of business structures. The classification system is based on three parameters: (1) how the company generates or creates value, (2) what stage of growth/maturity the company is at, and (3) what ownership and management structure the company has. (Dini et al. 2003, p. 17) In the interviews, the companies were asked those three questions and additionally some specifying questions when needed. Based on their answers, the classification of the companies can be seen in table 2.

Table 2. Classification of the interviewed companies.

Company	Value C4(M)	Value C4(S)	Value S1	Value S3	Growth	Personality
Company A		X	X		0	H2
Company B	X				0	H2
Company C			X		2	H2
Company D			X		2	L1
Company E	X		X		2	H2
Company F	X		X		2	H2
Company G			X		3	H2

The C4(M) as a value code means that the company manufactures intellectual products, in this case software, according to the company designs. The value code C4(S) means that the company makes software to customer specification. (Rathbone 2004) All of the companies fell into either of these two categories. Most companies also provided intellectual services, which are marked by value code S1. Intellectual services include for example consulting, designing and

educating. Growth code 0 means the company is a start-up company in a way that it is not completely profitable yet. Growth code 2 means the company is profitable and it has a one-level management. Growth code 3 means the company is profitable and it has more than one management levels. Personality H2 means a high-growth company with multiple private owners. Personality L1 reflects a one-owner private company that does not possess high growth characteristics.

6.2 Requirements Engineering Practices

6.2.1 Requirements Engineering Cycle

All of the companies studied work in an interactive collaboration with their customers concerning requirements engineering. How organized is the interaction, what methodologies are used, and who makes the decisions differ a bit from company to company. Next, the requirements engineering cycle practices, from eliciting the requirements to transforming feedback into requirements, are presented individually.

In company A, a vendor or a project manager elicits the requirements, as they are the ones working closest with the customer. The project manager composes a requirements specification which is a verbal description of the business / problem to be solved. The process continues with a functional specification of the requirements in which also UML is used. In addition, investment and cost calculations are made thereby. Another case of requirements eliciting and documenting in company A is an internal project. In that case, requirements are identified internally and are meant to increase efficiency and flexibility of operations. In both cases, documents are saved locally. Company A has many channels for receiving customer feedback: help desk, Internet portal, customer satisfaction inquiry and project managers, as they visit customers personally. Customer feedback is always refined inside company A and based on experience, a more intelligent way of implementing the requested update is proposed. Sometimes when customer requests a feature, but is not willing to pay for it, several same kind of requests from different customers are gathered together and marketed as one whole. Thus, the cost for a single customer is relatively smaller.

Company B product has its origin in the ideology of the company owners and the observation that there is no such product in the market. The product has been tested by potential customers and the feedback has given input for the re-implementation of the product in another way. However, the ideology of the product has not changed. The company B uses Resource Description

Framework (RDF) for modeling and designing requirements and stores documents locally. According to W3C (2004), RDF is a language for representing information about resources in the World Wide Web. RDF is intended for situations in which this information needs to be processed by applications, rather than being only displayed to people. Company B product is tested with potential customers that already understand the ideology of the product. Company B receives feedback in customer meetings face-to-face. Sometimes new features and updates are suggested based on another product the customers have seen somewhere else. Occasionally new ideas come up in informal discussions with other business partners and associates.

Company C elicits customer requirements in short meetings with customers. The requirements proposal is quickly composed during the same day and sent to the customers to be reviewed. After customer feedback the requirements proposal is refined. This loop is repeated until the customer is satisfied with the specification. Thus, company C is using an iterative requirements process. The requirements are then stored as tasks in a groupware tool, which is an internet-based software to ease working in a distributed company. Every task has an owner and a state, which is updated regularly as implementation proceeds. The tasks and their states are available to customers too. The requirements are specified in natural language, moreover, using UML and other methodologies. The groupware tool is being developed to include requirements management too. Feedback is normally received when implementation is ready and the software is in limited production use for one to three months. If customers do not give enough feedback, company C makes as many precise questions as needed to get it. Feedback is saved with the groupware tool. Customer responsible refines the feedback into software requirements and saves the requirements as tasks in the groupware.

Company D uses a free-form, interactive conversation in eliciting customer requirements. However, some big companies as customers obey a carefully predefined process in software development from eliciting requirements to implementation, integration and testing. Anyway, with most of the customers the requirements specification begins from small matters that eventually evolve bigger and bigger. The customers of company D do not demand modeling the general picture, describing the separate requirements is enough for them. The requirements are mostly documented in natural language, sometimes UML is used. Feedback is received most often in face-to-face discussion in Finland, sometimes through Internet with international customers. Feedback is very informal and needs to be refined. That is done in collaboration with the customer through iterative discussions.

Company E elicits requirements in interactive collaboration with the customers. Based on initial requirements the company makes a prototype, which is then developed according to customer feedback. Requirements are documented in natural language in the company intranet. Sometimes process models are pictured using UML, but not often as they change that quickly. The feedback company E gets is quite precise and needs very little refining. It is given in collaborative face-to-face meetings.

Company F has two ways of eliciting customer requirements: (1) asking customer information about the event structure and developing their product accordingly or (2) giving the company product to the customer for test use and then developing the product according to customer wishes. The company product is also developed internally according to customer feedback from several different events. Now requirements are documented in natural language in local storages, however, internet-based documentation will be taken into use in near future. Company F asks feedback before and after events, in which the company product is used. Feedback is given orally in telephone discussions and in emails or face to face in meetings. Often different customers want the same features. In addition, it often has been noticed internally too, thus, implementation is quite easy.

Company G has certain methodologies and models for eliciting requirements, however, it depends on the project, which is the most suitable one in each case. In general, the company uses the (Rational) Unified Process along with UML as a guideline. Projects, which have user interfaces (UI), often provide a UI-prototype. The prototype is then incremented in an iterative manner according to customer requirements. Requirements are elicited from customers in face-to-face interviews and documented as use cases and requirements specifications. Those documents are then updated iteratively and stored locally. Company G gets feedback from its customers in face-to-face meetings and in emails. Often many questions have to be asked in order to get enough feedback. The feedback is incremented in prototypes and documents.

6.2.2 Requirements Engineering Practices Summary

Requirements engineering practices in the interviewed companies were approached from two different angles: the requirements engineering process and the advanced state of requirements modeling. Requirements engineering processes used in the interviewed companies divided roughly into two groups: a free-form interactive process and a more predefined process. The requirements engineering process types in the interviewed companies can be seen in table 3.

Table 3. Summary of requirements engineering process types in interviewed companies.

Company / RE Process	Free-form interactive process	Predefined process and/or tools
Company A		X
Company B	X	
Company C		X
Company D	X	
Company E		X
Company F	X	
Company G		X

In the free-form interactive process, much of the work is performed in face-to-face meetings together with the customer, like eliciting the requirements, analyzing the requirements and receiving feedback. The predefined process has been defined, according to its name, in advance. In the predefined process, document templates, software tools, process models are examples of the tools that are used to ease the process and make it more consistent. In a way, it could be argued that the interactive process is predefined as well, if it is always used in a consistent manner. However, in this study, the examples of tools mentioned above are used to distinguish the two process types.

When comparing the software engineering and requirements engineering processes of the interviewed SMEs with the Rational Unified Process (RUP) presented earlier (in chapter 3.1), four observations can be pointed out. First, while software engineering still seemed mostly to obey the sequential development process, the iterative process has won some space in requirements engineering and it was used both in the free-form interactive process type and in the predefined process type. Secondly, it seemed that the bigger the amount of employees in a company the more structured and defined the requirements engineering process was. The biggest company in this study had the most advanced process based on principles presented in the third chapter. The second biggest company had a structured process as well. The two next companies were also categorized as having a predefined process, and the three smallest companies were classified to the group that had a free-form interactive process. Thirdly, the requirements engineering process in the interviewed companies is much more agile than the process described in the third chapter. It is also tailored to some extent according to the size of the customer, bigger customers obey their own more specified processes and expect their associates to obey them as well. Finally, management of requirements lifecycle was quite informal, only one company stated that it was planning to implement software that would include requirements lifecycle management.

The advanced state of requirements modeling divided into three classes: using natural language, using UML and meta-modeling the requirements. The results of the interviews can be seen in table 4.

Table 4. Summary of requirements modeling styles in interviewed companies.

Company / Requirements Modeling	Natural language	and UML	and Meta-modelling MDA, MOF
Company A	X	X	
Company B	X	(X)	
Company C	X	X	
Company D	X	(X)	
Company E	X	(X)	
Company F	X		
Company G	X	X	

All companies used natural language to describe the customer requirements. In addition, almost every company had experience in modeling the requirements using UML. In some companies, UML was not in every-day use though, that is why some of the X-marks are in parentheses in the above table. Again, it seemed that the bigger the company the more advanced requirements modeling it had in use. However, none of the companies had experience in meta-modeling, using for example MDA or MOF (introduced in chapter 3.2).

6.2.3 Decision Making

The decision making in the companies was studied in order to make comparisons with how different is decision making in proprietary software companies compared with open source projects. In the companies studied, decisions were mostly made more or less in collaboration with customers. The individual company statements next.

In company A, decisions about which requirements get to be implemented are based on requirement importance. Customer requirements are always more important than internal requirements. Interviewee A as development manager coordinates the changes and in which release they are implemented.

The company B management and development people together decide which requirements are implemented. The decisions are based on ease of implementation, the measure of requirement need, and gain to be expected. In

addition, the feature has to fit in the product ideology in order to be implemented.

Company C makes customers a proposal, which includes requirements to be implemented, schedule and the costs. Some customers want to have their say in the proposal but most are quite happy with it. The most active customers are produced pilot versions very early in the implementation phase, which are then commented.

In company D, the most critical requirements are implemented first. How critical requirements are, is determined by customers, when they are given the time it takes to implement the feature and the price it is going cost them.

Decisions in company E, are made in co-operation with the customer. However, since company product is rarely tailored, it is up to company personnel, the managing director and managers to decide in which order requirements are implemented. In addition, implementing the requirements takes such a little time, that the implementation order is of no importance to the customer. The requirements are sometimes not implemented as they would not gain the customer as much as they would make the software incompatible with future product versions.

In company F, customers decide which requirements they want implemented and which not. The decisions are made based on cost and schedule information given by company F. Often customers first want all of the requirements implemented but due to the schedule or monetary reasons all of the required changes may not be performed.

In company G, early prioritization of requirements happens already in use cases as they have a method for classification (mandatory, nice-to-have etc.). Eventually, it is always the customer, who makes the decisions. However, with smaller customers company G might give strong recommendations of the implementation because it has more expertise than the customer does.

6.2.4 Decision Making Summary

Decision making in the companies divided into three classes according to the decision maker: the customer, the customer and the company in collaboration, or the company alone. Although all the companies agreed that eventually the customer always makes the decisions, some points make the difference between these three classes. A company makes the decisions when the customer is only in a position where making suggestions to the software is

possible. A customer and a company make the decisions together, when the company has a very active role in the process. The company can for example give the customer strong recommendations for the solution. A customer makes the decisions when the role of the company is not especially active in the decision making process. The results of the study are presented in table 5.

Table 5. Summary of decision making in interviewed companies.

Company / Decision Making	Customer	Customer & Company	Company
Company A		X	
Company B			X
Company C		X	
Company D	X		
Company E		X	(X)
Company F	X		
Company G		X	

Normally customers make the decisions, based on timeline and work estimates given by software companies. That was the case with two companies. In the case of company B, the company makes the decisions itself based on their own ideas and possible suggestions from their potential customers. Most of the companies had an active role in decision-making together with the customer. Active role in these cases meant giving the customers advice, occasionally strong recommendations, on what would be the best possible solution for the customer, even if the best solution were not to implement the requirement at all. Company E also made independently decisions about implementation order and timelines.

The division of decision power between companies and their customers can be approached from the software production style point of view as well. Meaning, (1) if a company is selling software as a product, the decision power of a customer could be quite low in such case, and (2) when the company is making tailored software to customer specification, the decision power of the customer could be quite high in that case. When comparing those arguments to the information in table 2 (p. 65), some similarities can be noticed. Companies B, E, and F were named as companies that had a software product made to company design. As can be seen from the table 5 above, they are the companies with the most of decision power, thus, the cases are supporting argument number (1). On the other hand, company E is the opposite case of decision-making not supporting argument number (1) with customers having the decision power. The company E is tailoring their software according to customer needs quite much though, which could be the reason for this exception. Company D makes tailored software to customer needs and their customers make the decisions,

thus, that case supports argument number (2). On the other hand, rest of the companies that are manufacturing tailored software products and services are participating actively in the decision-making. Therefore, when software is tailored, not sold as a product, it seems the customer is not the only one making the decisions; the companies are taking a more active role in the process with their expertise.

6.2.5 Understanding the Language of Customer Domain

Understanding the language of customer domain was studied in order to find the possible needs and reasons for creating ontology. In general, the companies seemed quite satisfied with their situation concerning mutual language with their customers. Following are the results of how the SMEs and their customers manage to use a mutually understandable language.

Understanding the language of customer domain is not a problem in company A. Customer domain concepts and terms are known and the language of the customer is used in mutual discussions. Thus, software technological terms, for example, are not used with customers. The customer domain concepts are taught to new employees when familiarizing them to the company. The company A domain terms, however, have sometimes caused misunderstandings internally. Therefore, defining the concepts of company A domain is considered useful.

Company B has noticed that the term “knowledge intensive work”, is not very well known. Sometimes people are not even aware of the fact that they are doing knowledge intensive work. Since company B product is targeted to people doing knowledge intensive work, the knowledge gap has caused some displeasure in the company. Company B has defined the terms of their customer domain and is capable of speaking the same language with the customer. However, interviewee B notes that customers should define their concepts too.

Company C has had no language misunderstandings with their customers. The company aims to use simple language and always makes sure that the understanding of important things is mutual. Sometimes important terms and concepts are defined in contracts. Company C does not see any use in defining ontology, to work well, it should be extremely large. The value ontology could bring is too little compared with the amount of money defining it would cost. Interviewee C has worked in several countries and noticed that concepts have different meanings within a language, not to mention in different languages.

Company D has had no severe misunderstandings of terms and concepts with their customers, in principal. They have not defined their customer domain concepts but they do see formal definitions necessary.

Company E has a “rule” that technology-oriented employees do not discuss with customers. Therefore, the company has had no misunderstandings with customers. Employees with customer language skills also have technology language skills, so they are able to translate customer requirements into implementation requirements. Company E has defined some concepts of their customer domain and are in the process of deciding whether they should define them even more formally. The company sees defining customer domain concepts useful, especially in the open source field.

Company F is very satisfied with their situation. Interviewee F remembered only one occasion where the company had had a misunderstanding and that was not a severe situation at all. The company has not defined the terms of their customer domain and does not see it useful either.

Company G creates the mutual concepts and terms in co-operation with the customer during the requirements engineering process. Company G brings UML tools in the process and customers bring in their business knowledge. Company G has not officially defined the terms of its customer domain, but thinks it would probably be useful, if some third party did that.

6.2.6 Language and Ontology Summary

Language misunderstanding were divided into three classes according to the severity of the misunderstanding: major, minor, or no misunderstandings at all. Not a single company had had severe misunderstandings with their customers. Actually, most companies had experienced no misunderstandings at all. The misunderstandings that had happened were not critical. In some cases, the misunderstandings happened more inside the company than with customers. The results of the study are presented in table 6.

Table 6. Summary of having language misunderstandings with customers in interviewed companies.

Company / Language misunderstandings	Major	Minor	None
Company A		X	
Company B		X	
Company C			X
Company D		X	

Company E			X
Company F			X
Company G			X

Companies were also asked if they would see defining ontology useful. Answers were divided simply to yes, no, and maybe classes. The results of the study are below in table 7.

Table 7. Summary of the necessity of ontology in interviewed companies.

Company / Ontology useful	Yes	Maybe	No
Company A	X		
Company B	X		
Company C			X
Company D	X		
Company E	X		
Company F			X
Company G	X		

If the previous table is taken a look at, one could imagine that ontology would not be seen useful by the SMEs, but that was not the case. Despite of being quite satisfied with the mutual language with their customers, most companies stated that ontology would be useful to them. On the other hand, ontology is much more than a common language. It includes the concepts and their meanings, yes, but it includes the relations between concepts as well. Thus, ontology is needed for more than creating a common language. For example, when making new software, creating a model of concepts and their relations can be used as a basis for database and as a basis for the structure of software. Only two companies argued with the rest of the companies saying defining ontology for a certain domain could be a huge effort with no guarantees of satisfactory result that the results would not be worth the pay.

6.3 Co-operation with Open Source Community

6.3.1 Experiences and Expectations

Open source community co-operation experiences of the companies were studied in order to find out the background with which the companies would answer the rest of the open source software related questions. All companies had experiences of open source software, some companies actually worked

quite actively together with the open source community. The individual answers of the companies can be seen below.

Company A has used open source software as part of their product. They developed it further internally too. They have also given their requirements to open source community but at the end, they implemented the features themselves. Company A expects the co-operation with open source community to include discussing requirements about general features of the software. Requirements that are critical to company A, are not expected to be implemented by open source community. In addition, company A could give some part of their software to the community to be developed, if it needed renewal but was not timeline critical. In general, company A has retrieved good software from the open source community.

Company B has experience in using open source software development tools. Some open source software has also been integrated as part of company B product. Company B expects many challenges to be solved in the co-operation with the open source community. Those matters include strict limits of rights and responsibilities. In addition, further development of software, the right to sell the software and to what price, who owns the software and its idea, is there a guarantee and who is responsible for it, are matters concerning company B.

Company C has experienced co-operation with the open source community in many ways. It uses open source software, it has integrated open source software as part of the company software and it is developing software in the community as well. Company C expects that the collaboration with the open source community demands a lot of activity compared with proprietary co-operation. However, being actively involved with the community may also gain a company a lot.

Company D uses open source software as part of the company product. The company expects the co-operation with the open source community to be somewhat difficult concerning schedules and quality. The quality of the software may exceed your expectations or then again pass them under. The community releases the software whenever it wants, so the timeline cannot be predicted. Since the community is based on voluntary work, it produces the very features it actually needs. Company D thinks open source software is increasingly becoming a business.

Company E has many ways of collaboration with open source community. It uses open source software; it has integrated open source software as part of the company product and is actively involved in the community developing software. The company has even founded a Finnish user community of an open

source project. In addition, the company is planning to release software of their own under open source license. Company E expects the open source collaboration to have open communication; nothing can be hided. Interviewee E describes open source community as global, dictatorial management system.

Company F uses open source software as part of their product and develops it further internally. Expectations of co-operation are not very high. Communication problems might occur and the quality of open source software is not expected to be very high.

Company G uses open source software in software development and has integrated open source software into customer products. Co-operation with the open source community has been mostly presenting and discussing requirements. What company G is expecting from future collaboration with an open source project, is more openness in decision-making. Where the decision makers come from and in general, what is the future of the project are matters of interest to company G. Thus, company G would like open source projects to be a bit more predictable.

In the interviews, the companies were given three not exclusive alternatives from which they could choose. The same three alternatives, using open source software, developing open source software internally, and developing open source software with the open source community, are used below in table 8, where the results can be seen.

Table 8. Summary of open source software experiences in interviewed companies.

Company / OS Experiences	Use of OS	Internal Development of OS	OS Development with the community
Company A	X	X	X
Company B	X	X	
Company C	X	X	X
Company D	X	X	
Company E	X	X	X
Company F	X	X	
Company G	X	X	X

All companies had used open source software. In addition, all companies had either integrated open source software as part of their product and / or developed open source software internally further. Four companies had worked in collaboration with the open source community. Two of them had mostly communicated their requirements for the software they were using, but the

other two had participated in the implementation of open source software as well.

Companies were also asked to state their impressions and expectations of a possible collaboration with an open source project comparing with a proprietary project. Regardless of the fact that the companies were not given any alternatives from which they could choose and that this study was conducted for seven cases only, two arguments were mentioned more often than others were. The results are in the table 9 below. A company could name more than one expectation, which is why the total number is eight, not seven.

Table 9. Summary of collaboration expectations with open source community in interviewed companies.

Expectation	Numbers mentioned
Openness in decision making and communication	3
Unpredictable quality	2
Legal matters, contract matters	1
More active working	1
Collaboration among not-critical features	1

Almost half of the companies stated that decision-making in the open source community should be more open. It should be more obvious who makes the decisions, when is the software going to be released, what is included in the release, and in general what is the future of the project going to be like. In addition, open source community was seen quite difficult to communicate with, much of this is due to the fact, that a company needing something from the community has to be quite active itself, finding the information. Just as one company stated, collaboration with open source community demands a lot more activity from the company than regular proprietary collaboration. Yet, there is a possibility of gaining a lot as well.

Quality in the open source community was considered unpredictable. Some companies clearly argued that quality in the open source software is not as good as in proprietary software, however, in those cases the interviewee was not very experienced in open source software. Other companies stated that it is a matter of your own activity and skills whether you find quality software from the open source community or not. Anyway, both quality software and poor-quality software is known to be out there.

Collaboration with open source community was approached with some reservations. If a software requirement was critical to a company, a company

would prefer to implement it itself and not to do it in collaboration with the community. In addition, it was stated that it is still not very clear where the matters of contract law stand. Meaning, if a company collaborated with an open source project, who would have rights to sell the software, who would have to provide guarantees to the customer concerning quality and timeline, and who would have the maintaining responsibility of the software, just to mention some of the questions that came up in the interviews. Altogether, despite that there were doubts, from the company answers could be concluded that four companies had a positive approach to collaboration with the open course community and three companies had more or less negative expectations of it.

6.3.2 Knowledge of Open Source Software Development Characteristics

The case companies were quite well aware of the general characteristics of open source. Only one or two companies had disagreeing insights with the theories presented in this research. However, requirements engineering in open source community caused more opinions. Next, the arguments and the company answers are presented.

Most of the companies agreed that open source projects communicate through Web-based forums like websites, discussion forums and mailing lists. One company was not sure, but answered “apparently yes” anyway. Company E stated that there will also be face-to-face meetings in the near future; a project they are involved with is holding an international user summit in June 2004 and a Finnish Summit is being planned for the Autumn 2004.

Most companies knew that in an open source project there is a maintainer that has the decision power over the project. One company did not know the fact and one company answered “apparently yes”. It was also well known that traditionally power in an open source project has been achieved by contributing quality code. Company A noted that power is achieved either by working actively in the project or the company may be the founder of the project and thereby its maintainer. Company D added that providing quality code is not the only highly appreciated way of contributing to the open source project, good conversation and good comments are also appreciated within the community. Bigger open source projects are increasingly becoming like enterprises and need broader set of skills on areas like for example marketing and economics. Company E agreed with company D and added capability of innovation to the list of properties appreciated. Coding used to be highly valued, but it is not anymore.

Knowledge of who makes the timeline decisions in an open source project was also well known. One company did not know the fact and one company answered “apparently yes” to the question. Company D questioned the argument by asking if an open source project is ever officially released, since unofficial releases are made all the time. Company E noted that maintainers really do make the decisions about timelines and that timeline in an open source project changes a lot, usually for the later but not always. Timeline also depends on the size of the project.

Arguments concerning the phases of requirements engineering were not as well known as the general characteristics of open source development. Thus, more comments came up. Half of the companies agreed that open source projects do not officially perform business modeling. Rest of the companies were not sure, but did not deny it either. However, company D argued that big open source projects do perform formal business modeling.

Half of the companies agreed that in open source projects requirements are informally discussed and documented in Web-based archives. The other half answered “maybe” as they were not certain of the requirements practices in open source projects. Company E noted that requirements engineering is nowadays performed more carefully than some years ago. The maintainer controls the software versions and who is doing what, and to which version. Company D agreed with company E saying that formal requirements practices are used in big open source projects.

The argument about requirements analysis and design being informal, and performed as a by-product of implementation, made most companies hesitate. One company agreed without comments, but others had some reservations. Company A stated that there has to be formal analysis and design of requirements, for example in the Linux project. Company D noted that sometimes there are formal UML-designs in big open source projects. Company E however, commented that no reasonable open source implementer documents anything. Company G stated that some projects have pretty good documentation. However, whether the documentation was created before or after the implementation, was not known. In company G’s opinion, the documentation is not as good as in proprietary software though.

Last argument stated that if a company wants to gain decision power in an open source project, it has to participate actively in the implementation phase as well. Two companies had no knowledge about if this was the practice in open source community or not. One company agreed without a doubt, and the rest of the companies had some observations. Company A stated that good arguments are a way of influencing the decision-making of an open source project. Company D

noted that donating money, good conversation and comments are also ways of gaining power. Company E said that participation in the implementation phase is not as important as it was earlier, other activity and innovations are becoming increasingly important nowadays. Company G agreed that donating other resources (e.g. money) for the project is an alternative way of gaining power.

Summary of the knowledge of open source development characteristics can be see in table 10.

Table 10. Knowledge of open source development characteristics.

Interview Argument	Yes	Conditional yes Maybe	No	Do not know
Web-based communication	6	1		
Maintainer has the power	5	1		1
Power achieved by contributing quality code	2	4		1
Timeline decisions made by a maintainer	4	2		1
No Business Modeling	4	3		
Informal requirements	4	3		
Informal analysis & design, by-product of implementation	1	6		
Implementation activity counts in gaining decision power	1	4		2

When comparing the information in table 10 to the information companies provided when asked about open source collaboration expectations, it can be noticed that the same issues came up. Decision power and ways of achieving it were the arguments that caused most of the comments. Mostly companies argued with literature about ways of achieving power in an open source project. Quality code and participating in the implementation are not as important as they used to be, active conversation, quality comments and innovations, and donating money are other ways of gaining respect and power in the open source community.

Analysis and design being informal by-product of implementation phase caused many arguments as well. The companies stated that in some open source projects, the documentation is very good, and usually the project in that case is a bigger one. However, it was not known whether the documents had been produced before or after the implementation. The analysis and design

documents would be most valuable if they were produced before the implementation.

6.3.3 Acceptance of Open Source Software Development Characteristics

Next, the companies were asked if they would accept the characteristics of open source development when collaborating with an open source project. The same arguments as in the previous chapter were used. This time a lot more comments came up. In addition, these arguments dig up “no” answers, which were completely missing in the previous chapter.

Web-based communication forum was quite acceptable to all companies. However, company B noted that whatever the forum will be, it should not be public for the entire world to see. In addition, a company or equivalent should not own it. Instead, an independent third party should provide it. Company F stated that face-to-face meetings would be necessary at least in strategically important phases of the project. Company E stated that web-based communication is sufficient in the beginning. In their opinion, there should be from 100 to 500 000 users in a project in order for face-to-face meetings to be beneficial. Company G agreed with company E, a web-based forum is good for basic communication, but not comprehensive. Company G added that there should be proper software doing the job; a plain web site is not enough.

Decision power and ways of gaining it caused comments the most. The thought of a single maintainer making all the decisions was not acceptable to two companies. Three companies stated that power has to be gained otherwise than by active working in the project. Almost all companies had strong opinions on the arguments. Company A stated that if the collaboration is not significant to their business the maintainer making the decisions is acceptable. Otherwise, criteria of how decisions are made and how companies can participate should be formed. Power should be achieved also by some other way than by making quality code, good ideas should be taken into account anyway. Company B answered that power should be attached to costs; if a company is not paying anything; it should not have power either. In general, interviewee B stated that there should be conversation when needed and it would be recommendable that the company, which gave the requirement, had some power over it. Company B also stated that small companies do not have resources to work actively in open source projects, therefore, it is their fear that big companies will “steal” open source projects and eventually introduce similar proprietary software products. Company C said it is completely acceptable, that the maintainer makes the decisions, after all, that is the way open source projects

work. The company does not object active working in an open source project. The point is that with a small contribution, the company can gain a lot and that eventually, there has to be paying customer. Company C also said if you are able to give good reasons for why a feature is needed, you might get it accepted. Company D stated that the power has to be gained by active working and resources, people or money. Alternatively, the decision-making organization could be like RosettaNet, an open organization. Company F does not accept the maintainer as a single decision maker. There has to be a way to influence an open source project, especially if one has a requirement for it. Company F thought it is in general fair that those who have worked actively in the project have the power. However, active participation is very difficult for an SME because of their small resources, people and monetary. They suggest considering alternative decision making reasons, for example voting. Then, first a decision should be made how many votes do companies have, for example do they have one vote per company or are companies given the more votes the bigger the company is. Company G stated that there has to be a group, which makes the decisions. However, it is very important that the decision-making process and the decisions are transparent. Company G added that the right to participate should be given to active companies because of their good proposals; implementation should not be needed.

The argument that the maintainer makes the decisions about the timelines caused some comments too. Company A said if the software version to be was business critical, then it was not acceptable not to be able to influence the timeline decision; otherwise, it was okay. Company B stated that decision power should belong to the party who is responsible for maintaining and marketing the software. In general, they thought timelines to be a big problem. Company C had a different approach. It said the collaboration with an open source project could not be built on future versions of open source software. The open source product has to be taken as it is. With this mentality, timelines will not be a problem. Company E stated that the only thing bothering it concerning timelines (in their present collaboration with the open source community), was the making of a local language version of the software. That should be given to a company who has a genuine business interest in completing the job. Company G requested the timelines and included features to be more predictable than they now are. For example, in a certain open source project, a group of experts makes a proposal of features, which is then commented in a certain timeline. That kind of style could also be used in the DBE. Anyway, if open source software does not have the feature, the company G implements it itself or acquires it from somewhere else.

The companies did not long for business modeling, surprisingly. Company A was willing to give their business modeling outputs to the community. Company

B agreed but needed the information and guarantee about when the modeled business requirements would be completed. Company F agreed with company B. Company C added that if the business modeling output given to the open source project were viable it would evolve in the community.

Informal requirements documentation, analysis and design were not problems to the companies interviewed. Company A only needed interface-type of documentation to exist after the requirement had been implemented. The company thought it was more important to get a feature that was working well than to get fancy documentation. Company C noted that actually, most open source projects wish there were people involved who would like to write more formal documentation. Company D did not need formal modeling of requirements, instead, it needed to know that the requirement has been noticed, how it is evolving (or is it), and when is it going to be ready to use. Company D would like open source projects to work in a formal manner, if it was participating in the work. Company E noted that open source projects develop features from general level to specific, and that is exactly the opposite way of working to proprietary software world. Company F also needed information about how requirements are evolving in an open source project, especially if the requirement was given by them. Company F is not willing to participate in an open source project at any level because of their limited resources. Company G may give informal requirements for the community for further development. In general, company G notes that open source projects should manage the requirements more formally. Company G can do analysis and design documents for a feature they proposed themselves. In general, architecture and interface documents are the ones that matter, others are not that important.

Participating in the implementation phase of an open source project in order to gain decision power did not directly appeal the interviewed companies. Company A stated that they are willing to give feedback and comments to the community, and that should be enough. If the requirement were critical to their business, company A might also provide a person to the implementation phase in order to fasten the timeline. Company B said participating in implementation is impossible for a SME because of limited resources. Company C however, stated that if participating was business-wise profitable, then giving their input into implementation phase is acceptable. Company D stated that contributing into implementation phase is not an only option to gain power in an open source project, donating money and generally working actively for the project are ways of gaining power too. Company E noted that they are willing to participate in the implementation, but that would happen only after the project brought some money in the company. Company F did not like the idea of gaining power by implementing quality code or donating money. It stated that there should be some other way to achieve a high decision-making position, for example

arranging face-to face meetings. Company G could participate in implementing open source software if it truly saw advantages in participating in the phase.

Acceptance of open source development characteristics summarized can be seen in table 11.

Table11. Acceptance of open source development characteristics.

Interview Argument	Yes	Conditional yes Maybe	No
Web-based communication	4	3	
Maintainer has the power	2	3	2
Power achieved by contributing quality code	1	2	4
Timeline decisions made by a maintainer	1	5	1
No Business Modeling	4	3	
Informal requirements	1	5	1
Informal analysis & design, by-product of implementation	4	2	1
Implementation activity counts in gaining decision power		5	2

As can be noticed from the above table, once again, the companies expressed their need for decision power and possibility to influence the decision making of an open source project. Majority of companies considered unacceptable that in open source projects decisions concerning requirements and timelines are made by maintainers, no one else. This notion is not surprising at all, when comparing it to the results of decision-making question (in chapter 6.2.4). There it was noticed that customers are not anymore the only ones making the decisions; in addition, companies are actively involved in the process. Thus, when the companies have gained decision power in their business, it is for sure a strange idea that they should let someone else make all the decisions.

6.3.4 Additional Comments and Ideas of Co-operation

At the end of interviews, companies were given a chance to state freely their opinions of open source collaboration; in case the questions had not covered everything, they had in mind. Following are the individual comments of companies.

Company B argued that open source software will not able to compete against proprietary software. Its credibility, timelines, responsibilities, guarantees, rights

and maintaining matters are too incoherent at the time. The community cannot provide a 24/7 service. In general, company B needed the future features and timelines to be announced in advance.

Company C thought the biggest problem for the DBE project would be getting enough companies involved. The project has to come up with usable business cases and genuine short-term economical gain in order to make the project desirable for the SMEs. In addition, technological architecture of the DBE has to be clarified, in beginning of 2004, is was still quite confusing.

Company E would fasten the timeline of releasing the DBE software into the open source community. As it is, the DBE seems more like a proprietary project than an open source project. Company E has doubts about how committed developers can be elicited to the project after all the interesting work has already been done. A small pilot version of the DBE platform, not the finished version of it, should be given to the community to be further developed sooner than the plan now is.

Company F would like open source projects to communicate better to the proprietary world. A mechanism of giving feedback to the project would be needed and information from the project to the SMEs about timelines and possibilities of implementing software components would be needed as well. In general, company F stated that SMEs need to have some power when collaborating with open source projects, but their resources (time, personnel) are limited, thus, the mechanism of co-operation should consider that.

Company G has good experiences of the quality and quantity of software in the open source community. The company states however, that an open source project should not be too organized, or it might lose some of its speed and flexibility.

7 Conclusions

The main research problem of the thesis was: what information flow and information management practices are needed when the small and medium sized software enterprises are developing software services for the DBE? First, existing theories were examined to describe the background of the research. Then, the knowledge found was utilized in examining empiric cases. Conceptual approach was applied in the theoretical part of the study, in which general understanding about requirements engineering collaboration between proprietary companies and open source projects was established. The sources of information used in the theoretical part were: books, scientific articles, Object Management Group (OMG) specifications and Internet pages. The Internet had a quite important role in collecting information about open source software, because it was difficult to find books and scientific articles of that topic. Although that kind of sources of information is not recommendable in the scientific writing, as they are not proven scientifically, they cannot, however, be neglected when it comes to open source software, since the open source software community is a very important source of information regarding itself. In the empirical part, action-oriented research approach was applied. Case method was used for examining seven companies. All of them were proprietary software producers, which gave an opportunity to examine their requirements engineering practices. Most of them were actively working with the open source community, which made it possible to study the point of view of open source software collaboration as well. The requirements engineering collaboration between proprietary companies and open source projects was examined by interviewing the companies.

In order to answer the main research question, four sub-objectives (Chapter 1.3, p. 5) were set. As both the theoretical background and the experiences of the companies have been examined, the sub-objectives can now be answered. The first sub-objective was: why is ontology needed by the SMEs and how is it used? In the chapter 2.4, four benefits of ontology were stated based on theories presented earlier in the same chapter. First, ontology offers an opportunity of lessening diversity in commonly dealt information. Secondly, it reduces errors and noise in information leading to saving time and money. Thirdly, ontology facilitates better co-operation both within and across organizational boundaries. Fourthly, it offers the basis for optimizing organizational change by the right set of tags. When comparing these arguments to the company answers, a few observations can be pointed out. Most companies saw defining ontology useful, however, it was difficult to get answers to the question why ontology could be useful, even from the companies that did consider ontology useful. Therefore, the following results are

only impressions of the interviewer. Good reasons for defining ontology could be lessening diversity in information and reduction of error and noise. However, that impression might have been caused by the previous question in the interviews, which handled language and concept misunderstandings with the customers. As already stated in chapter 6.3.2, ontology consists not only of concepts, it includes the relations between concepts as well, which allows the concepts to be subcategorized according to their essential qualities. Thus, the question handling only the concepts and language with customers might have led the interviewees to think too narrowly. On the other hand, not a single company saw ontology as a way of managing organizational change by suitable tagging or as a way of facilitating better co-operation. If the companies were experienced with ontology and the background theories, they would quite probably have stated their exquisite opinions. As it is, only one interviewee gave good grounds for his opinion, which was one of the two negative attitudes towards ontology. Therefore, ontology and the opportunities it yields do not seem quite familiar to the interviewed software producing SMEs yet. It can only be speculated that ontology is possibly needed by the SMEs to reduce errors and noise leading to saving time and money, and facilitating better co-operation within and across organizational boundaries.

The second sub-objective was: How do software requirements engineering principles in practice support managing the information needed by the SMEs? Information management can be viewed as the management of network of processes that acquire, create, organize, distribute and use information as a continuous cycle. Requirements engineering, then again, includes elicitation, analysis, specification, verification and management of software requirements. The goal of information management is to transfigure information into learning, insight, and commitment to action, whereas requirements engineering aims to transfigure information first into learning and insight, then into action, the implementation of the software. Consequently, requirements engineering is a way of managing information. Six closely related processes for information management were already suggested (in chapter 2.3.1) by Choo (1998, p. 260): (1) identification of information needs; (2) information acquisition; (3) information organization and storage; (4) development of information products and services; (5) information distribution; and (6) information use. Requirements engineering principles mostly support the first four processes of Choo's information management model, thus, they are covered below.

Choo (1998, p. 262) stated earlier (in chapter 2.3.1) that information needs are discovered from problems, questions, and ambiguities through different situations and experiences in an organization. The information has to be rendered meaningful to certain individuals in certain situations; thus, the meaning of information is not the only thing concerned. Requirements

engineering begins with analyzing the problem. In the first step of requirements workflow (introduced in chapter 3.1.3), the problem is analyzed and the aim is to gain an agreement on the statement of the problem to be solved. In addition, the first step of requirements workflow includes identification of stakeholders, persons who have a stake in the project outcome, for example different sets of users. The stakeholder needs are elicited and captured in the second step of requirements workflow. The needs are captured as requirements with the help of use cases that are modeled with UML. The use cases are a method of describing what each set of users wants the system to do for them. Each set of users also produces several different use cases for different situations of using the system. Thus, the system requirements information described by use cases is meaningful for certain individuals in certain situations, just as Choo above insisted.

When comparing the just presented theoretical background to the results of the SME interviews, the requirements workflow is quite similar, yet, more simple in some cases. The stakeholders are simply the customer companies negotiating with the SMEs. Whether a more thorough identification of stakeholders is performed within organizations of the customers or not, did not come up in the company answers. The capturing of the requirements is performed with the help of UML, sometimes yes, but more often in natural language. Traditional requirements specification document seemed to have sustained its popularity as a starting point of capturing requirements. Business modeling (introduced in chapter 3.1.1) was not mentioned as a separate phase of requirements capturing. However, that does not mean it would not be performed, it could just be performed as an integrated part of requirements capturing. It seemed that the size of the SME and the size of the customer are influencing matters regarding how the requirements are modeled and documented. Namely, the bigger the company, the more predefined requirements process it possessed and the more developed requirements modeling it used. This does not mean, however, that smaller SMEs would perform their work worse than bigger ones. Smaller companies often carry out smaller projects, which do not need to be as structured as bigger projects. In addition, both big and small companies tailored their processes and methodologies according to needs of bigger customers, when required.

Choo (1998, p. 262) also stated (in chapter 2.3.1) that information acquisition sources have to be planned for, monitored, and evaluated just like any other resource of the organization, but he admitted it to be a complex task to do. On one hand, the information needs are wide ranging and on the other hand, human attention and capacity are limited. Choo (1998, p. 264) continued that the ability to absorb to variety can be improved in many ways: using information professionals, outsourcing specific issues, and using information technology,

yet, people are the most valuable information sources in any organization. That is the case when acquiring information for software requirements as well. In many cases, technology, in the form of current software, is the basis from which the requirements for new software can be elicited. Yet, that is not enough. Humans are capable of analyzing the problems in current software and turning that information into requirements of new software. If every single person that is to use the new system were asked an opinion, a complex situation to manage would be caused. In requirements engineering, the variety of information sources is managed by selecting typically one person per a group of stakeholders that have similar job descriptions, to give the group requirements as use cases or in natural language.

In the interviews, the SMEs agreed that humans are the most important information source. Requirements are elicited, captured, and analyzed often in face-to-face meetings in collaboration between the company and the customer personnel. Occasionally, old software is not needed, just the opposite, it is just in the way of innovative creation of new solutions. On the other hand, new ideas can sometimes be caused by another software product, for instance by a competing software product. Anyway, in requirements engineering information is always acquired by humans from humans, with or without using current or competing software or other information technology in assistance.

Requirements information organization is performed according to software functions, for example requirements for a user interface, requirements for a database table, and requirements for a background program. Often a requirement influences many places, for example functionality in a customer information user interface may affect the user interface, a background program and a customer information database. In that case, separate requirements should be created and linked together via requirements management tool or equivalent. Then, the three different requirements would not be forgotten and in case any of them changed, the possible need to change the other two would be noticed as well. The requirements information storing can be carried out in multiple ways depending on the company practice. Traditionally requirements have been gathered in a requirements specification, which is a document written in natural language. This was the case with the interviewed companies as well. The document may also contain preliminary pictures of the user interface and preliminary models of the database and the software architecture. Use cases, both in natural language and in UML, have been gaining popularity recently as a way of describing and storing the requirements. Most interviewed companies used UML too; yet, it was not the obvious choice for most of them. Only one company, the biggest one, stated that modeling the requirements with UML is more a rule than an exception. Often use cases are an addition to the requirements specification in order to verify the working of certain key features

of the software system. Sometimes, requirements are stored individually in a database with a software product meant for the purpose. This was the case in one company. Every requirement is given for instance a title, description, state, person responsible, timeline and most importantly, a link to other requirements that may be due to change if this requirement changes. With the help of these software products, the lifecycle and change management of requirements can be made easier. Recently Internet-based documentation has gained popularity but most of the interviewed companies had their documents in local storages, although plans for taking Internet-based documentation into use had been made in few companies.

Altogether, compared with the theories presented in the second and third chapter, the requirements engineering processes and methodologies supporting the information management of the interviewed SMEs seemed to be quite agile and tailored according to the size of the project and sometimes the process of the customer. As a main rule, requirements were collaboratively elicited and analyzed with customers and documented as a requirements specification in natural language. Requirements were not always modeled with UML as the models change that quickly and UML modeling is more time consuming than using natural language. Meta-modeling the software system with MDA and MOF (presented in chapter 3.2) was not familiar to the SMEs. If meta-modeling had been familiar to the companies, it would probably still not be used regardless of its advantages (chapter 3.2.1, p. 3x), as it is even more time consuming than using UML is. It seemed that bigger SMEs obeyed more carefully requirements engineering processes and methodologies. However, even smaller SMEs tailored their requirements processes and methodologies when needed according to the process of the customer, as bigger customers would expect them to obey their own, more developed processes.

The third sub-objective was: *What are the ways of working with open source software projects for an SME?* In this study, it was found that SMEs could not be treated as one group, as their experiences, expectations, and attitudes regarding open source software, are not uniform. Therefore, the seven interviewed SMEs were classified into three groups regarding their collaboration with open source community: strategic, active, and observing collaborators.

1. *Strategic collaboration* with the open source community indicates that the company business idea is somehow based on open source software. For instance, a company can offer different kinds of services to open source software product in addition of selling it. Strategic collaboration also indicates that the company is actively involved in the community and understands the basic idea of the open source community: if you take something from the community, it would be advisable to give something back. Additionally strategic collaborators possess enthusiastic attitude

towards open source software. Two companies out of seven fell into this category.

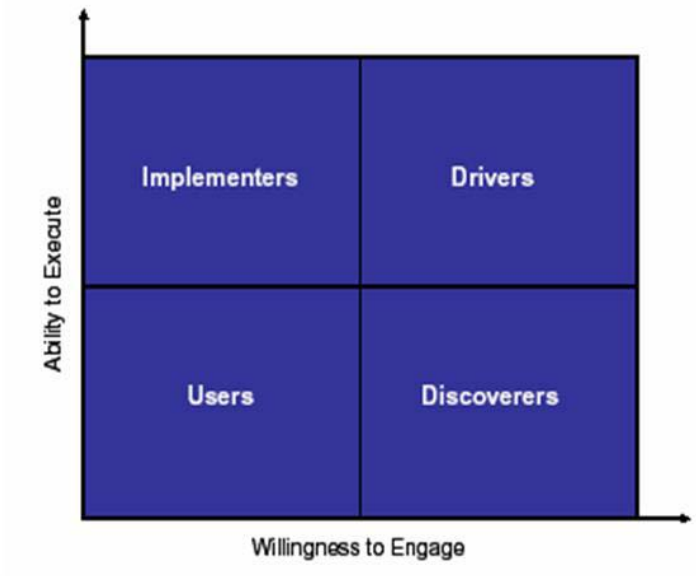
2. *Active collaboration* with the open source community means that the company uses open source software as part of the company product or in the development of it, but the open source software use is not especially strategic to the company. The business idea of the company is not based on open source software. Active collaborators might give their requirements to the open source community, but would probably not participate in implementing open source software. In general, active collaborators have a positive attitude towards open source software. Three companies out of seven fell into this category.
3. *Observing collaboration* with the open source community indicates that the company is using open source software, typically software development tools made by the community, thus, software is taken from the community, but rarely anything is given back. The attitude towards open source community might even be a bit negative in observing collaborator's case. Two companies out of seven fell into this category.

The division of companies into the three groups conformed the experience level of open source software of the interviewee. The least experienced interviewees came from observantly collaborating companies. The most experienced ones came from strategically collaborating companies and rest of the companies were active collaborators. The division conformed the favorableness towards open source software as well. Which one came first, the positive attitude towards open source software or the experiences of collaboration with the open source community was not examined in this study. However noticeable is, that within the interviewed companies, the more experience the company had from collaboration with the community, the more positive attitude towards the community it had. Therefore, if the positive attitude existed first, it had not changed with open source community collaboration.

This classification of open source collaboration can be compared to a division of open source software use by Heikinheimo (2003a, p. 10). She has presented that using open source software in companies can be divided into four groups: internal use, product use, shaping market situation with open source software, and open source software as a business. When comparing Heikinheimo's division with the division presented in this research, some similarities and some differences can be found. One of the companies, categorized as an observing collaborator in this study, is using open source software internally; the other observing collaborator has integrated it as part of their product. Active collaborators could be described either to use open source software in products or to shape the market situation with it. All active collaborators in this study belonged to the group that uses open source software in their products.

Strategic collaborators belong to group that has developed a business of open source software. Altogether, in this study, two different types of open source software use, internal use and product use, were classified as the same type of open source software collaboration, observing collaboration. Therefore, when comparing the two divisions, they cannot be described as identical. Although both classifications considered open source software use, additionally the collaboration division presented in this research, considered the importance of open source software in the company business, attitude towards the community, and the balance between taking from the community and giving to the community.

Another interesting classification of SMEs is that of the DBE project itself. In the DBE training work package, the SMEs have been grouped in four clusters: drivers, discoverers, implementers, and users. The SMEs were characterized by two dimensions: their technological and behavioral ability to execute the DBE and their willingness to engage in the DBE. (Anon 2004, p. 16) The classification can be seen in picture 19.



Picture 19. The DBE categorization of SMEs (adapted from Anon 2004, p.16).

Drivers have strong capabilities and interest to participate in the DBE at a very early stage. With a profound background in the relevant technological and business concepts, they can absorb the necessary DBE knowledge faster than other participants can. (Anon 2004, p. 18) When compared to the classification presented in this study (p. 90), strategic collaborators could be described to possess quite the same characteristics as drivers. However, the DBE division is not emphasizing the nature of open source software use of the company as

much as the division in this study. Therefore, in addition to the two strategic collaborators, there is one more company that could be classified as a driver, thus the number is altogether three. Implementers have similar technological capabilities as drivers that allow them to develop components and contribute to the DBE. However, this larger fraction of developer SME is expected to show a less steep learning curve and reduced engagement when it comes to building the DBE community than the drivers. (Anon 2004, p.18) Implementers could be described as active collaborators, when compared to classification presented earlier (p. 90). Thus, there were two of them in this study. Discoverers have limited technological capabilities and other business interests than technology development. Therefore, they are mainly able to use DBE services and are not expected to contribute with components. However, they are strongly engaged and are expected to see a significant individual case in using the DBE. (Anon 2004, p.18) Discoverers are a group that no SME was classified to in this research. Accordingly, it is no wonder that the category was not found either. However, the category is important and needs to be taken into account when planning training activities of the DBE project. Users have similar initial capabilities to discoverers. However, they are expected to be more focused on individual application and personal benefits and be mainly passive users of the community. (Anon 2004, p.19) This category is comparable to the observing collaborator group presented earlier (p. 90). Thus, two SMEs can be classified in this category. Altogether, the classification presented in the DBE training work package and the one presented in this study seem to be quite identical. Although the reasons for different categories were a bit different, the outcomes are still the same. The DBE classification is a bit more accurate, measured both by number of characteristics given to each category and by the number of categories. Additionally, the DBE is the case project of this study. Therefore, in this study the DBE classification is used from now on.

The fourth and last sub-objective was: *What is the requirements engineering recommendation for the DBE?* In chapter 3.1.2, business modeling was described as a way of identifying the business problem to be solved with software, and its results as an important source of the first requirements for software. In chapter 4.4.4, according to the references studied and found for this research, it was argued that business modeling is simply not performed in the open source community. In the interviews, it was found out that the SMEs do not perform business modeling either; instead, they use the traditional requirements specification as a starting point for capturing requirements. Thus, the SMEs did not long for open source projects to perform official business modeling. In case the SMEs had requirements of their own for open source software, they were willing to perform business modeling themselves when needed, and to give the results to the open source project to be further developed. However, the companies needed the open source project to

communicate back as well. The open source project should someway inform the original producer of a requirement of how is the requirement evolving and when it is going to be published.

In chapter 3.1.1, requirements capturing, analysis, and design were named as the next phases of requirements engineering. The capturing phase includes listing candidate requirements, understanding the system context and capturing both functional and non-functional requirements, which are then refined in analysis and design phases for the implementation of software. In chapter 4.4.5, it was argued that in open source projects requirements are more informally discussed and modeled than in proprietary software companies. However, the interviewed companies had requirements engineering practices that were not as far from open source practices as expected according to the theoretical part of this study. It seemed that the SMEs had quite agile requirements practices, which were suitable considering the size of the companies and the size of their customer projects. Additionally, all companies tailored their ways of working if needed in case of a bigger customer that requested a more formal process. What is formal enough then? The companies stated that documentation in open source software projects, in general, is quite good, although it is not as thorough as proprietary software documentation. Even some UML-models had been found, at least in some bigger open source projects. The most important documentation, the companies thought open source software projects should always provide, is a description of the software architecture and interfaces. Again, the companies were willing to give their requirements to open source projects, and even to perform the analysis and design phases as well, in case they considered it beneficial and they thought the open source project way of performing the job was too informal. However, once again, the SMEs needed open source projects to inform the evolvement of requirements someway back to the requirements origin. Additionally, managing the requirements more openly and clearly was requested to ease independent information searching of open source projects.

Requirements management, decision-making on requirements priority and changes (p. 33) is performed throughout the requirements engineering process. Usually customers make the decisions, based on timeline and work estimates given by software companies. In the interviews, the active role of software producing companies came up. Active role in these cases meant giving the customers advice, occasionally strong recommendations, on what would be the best possible solution for the customer, even if the best solution were not to implement the requirement at all. Additionally, the companies transformed the feedback given by customers into software requirements, only few companies stated that their feedback is refined enough to be implemented into software as such. Companies made independently decisions about implementation order

and timelines as well. Thus, the role of customer as a sole decision-maker is not as strong as it used to be, the companies have taken a more active role in the process with their expertise. When comparing that development to the fact, that open source projects are dictatorial communities, where a maintainer or a maintainer group makes all the decisions, a conflict situation could be near. The SMEs do not accept an arrangement like that at all, which came up several times in the interviews. When collaborating with open source projects, the SMEs expect to be taken into account because of their general activity, good ideas, and documents. That should be enough for gaining respect and decision power over an open source project, participating in implementing the software, which usually has been nominated as the way of gaining power (chapter 4.3.1), should not be needed.

Now that all the sub-objectives have been answered, the research problem of this study, *what information flow and information management practices are needed when the small and medium sized software enterprises are developing software services for the DBE*, can now be answered. Before that however, some principles according to which the question is answered, have to be presented. First, the DBE is going to be an open source project in three years, but it is not one yet. Therefore, it cannot automatically be assumed that all the open source characteristics presented in this study, would apply to the DBE as well. On the other hand, the fact that the DBE is not yet an open source project is an opportunity. The future ways of working as an open source project may be planned and possible developed according to SME needs. Secondly, the DBE is now a large project, thus, it is likely to be a large open source project as well. Therefore, the characteristics of large open source projects are likely to be the basis of DBE open source project. Finally, the SMEs were classified into four groups according to their ability and willingness of collaboration with the DBE: drivers, implementers, discoverers, and users. The expectations and needs of information flow and information management practices within these three groups are a bit different. Therefore, although the research problem is answered mostly concerning the SMEs as one group, some aspects are explained more thoroughly group by group.

The communication facilities of any project are of major importance. The collaboration between the DBE and proprietary SMEs does not make an exception. Traditionally open source projects have communicated via Web-based forums, like web sites, discussion forums, and mailing lists. According to the SMEs, the web-based forums are acceptable for the collaboration with the DBE; however, some adjustments can be stated by covering each collaboration group individually. First, drivers have made a business of open source software and they are actively involved in open source projects. Thus, they are experienced web-based forum users. Additionally, they are experienced in

searching information of projects independently. Therefore, drivers might be satisfied with the situation as it is. Implementers have some experience in using the web-based communication forums of open source projects as well, but since it is not their actual business, they are not experts in it. Implementers need the DBE to reorganize the traditional open source way of communicating in the Web. Information has to be more organized and structured as it now is to make information searching easier and the project way of working more formal. Users agree with implementers on the need of reorganization and restructuring of web-forums, but in addition, they want more. Users needed face-to-face meetings to be arranged at least in strategically important phases of the DBE. Altogether, the communication arrangements should enable fluent information flows both from SMEs to the DBE and from the DBE to the SMEs.

The future decision making principles of the DBE project are even more important than communication practices. Traditionally a single maintainer or a group of maintainers have made all the decisions in open source projects. People, who have wanted to participate in decision making, have had no other way to gain decision power than contributing quality software code and preferably often. If the SMEs were collaborating with the DBE open source project, those characteristics can, once again, be considered via those four SME categories presented earlier. Drivers know how open source projects work; the decision-making hierarchy of open source projects is not a problem to them. They are also willing to participate in the implementation phase if it is seen economically beneficial. Implementers are not as approving. They want to be taken into account because of their good ideas and requirements. Implementation of software should not be needed to gain decision power. The power should be based on other activity performed for the project or some other solution, for example an open organization, could be formed. Users do not accept the traditional decision making ways of open source projects at all. Their arguments are the same as implementers had. Additionally, users emphasized the limited resources of SMEs, they do not have personnel to participate in open source software implementation and donating money in order to achieve decision power is out of the question as well. Altogether, the DBE project should notice that SMEs especially need to participate in decision-making concerning software requirements they have requested from the DBE.

In this study, based on the theoretical part, a gap between proprietary companies and open source projects, concerning the formality of software requirements modeling and documentation, was expected. However, it barely existed, nicely working software, produced by open source projects, was more appreciated than formal documentation. Usually, when SMEs are collaborating with an open source project, information flows from an SME to an open source project. The information may consist of for example requirements, comments,

and in general, analytical discussion. Those principles were quite acceptable to the SMEs. The drivers and implementers, even one of the users, were willing to model their requirements themselves and pass the information on to the DBE, if they considered the DBE way of modeling requirements too informal. However, that might not be the case, as the DBE obeys modeling and meta-modeling practices that are of high standards. Actually, the need to increase the formality of requirements modeling might concern more the SMEs than the DBE. That would especially be the case if an SME had legacy systems that it needed to be modeled in the DBE and the SME would not be used to perform structural modeling with for example UML or MOF.

Traditionally, an open source project communicates back by putting information to be visible in the project web site or equivalent. However, the SMEs needed the DBE to do much more than that. The SMEs, especially implementers and users, wanted the DBE to produce information about that their requirement has been noticed, how the requirement is evolving, and what comments has it received. The DBE should also inform the SMEs about which requirements are going to be included in which release and when are the next versions going to be released. Additionally, some basic knowledge about the open source software licenses, rights and responsibilities could lift especially the enthusiasm of users, which might be not very high partially because lack of appropriate knowledge.

Information management practices are in an important role, when wishing for a successful collaboration between the DBE and the SMEs. What kinds of information management practices are needed and when, depends quite a lot on what kind of an SME is at issue. Drivers already possess the enthusiasm and abilities to engage in the DBE. Therefore, to maximize the possibility of successful initiation phase, drivers should be the first group of SMEs to collaborate with the DBE. Drivers should be informed about the DBE in general, but especially about the business potentials and practices, the DBE generates. The drivers can also be the first companies to implement services in the DBE, thus, information about the DBE service development practices would be appropriate. In addition of drivers, discoverers should be taken into account in the initiation phase as well. The discoverers might not possess technical abilities, but their willingness to participate in the DBE is more important. Their information needs are quite the same as the needs of drivers were, basic DBE information and information about the business possibilities. In the second phase of the DBE project, the implementers and users should be encouraged to engage in the DBE. As the implementers possess more abilities to engage in the DBE, they should be provided with information about DBE services development, in addition of basic DBE introduction. Users do not possess the high will nor any special abilities to engage in the DBE, but they are the biggest

group of SMEs (Anon 2004). Therefore, engaging them is important in order for the DBE to succeed widely. However, the users should be approached carefully, not too early and not with too much confused information. In the second phase, basic introduction of the DBE could be enough. The users might be easier to engage in the DBE, if there was not so much time, effort, and learning needed. Another organization, for example a Regional Catalyst like the TTC, could operate as an intermediary, providing services and education, which would lower the threshold of the users of engaging in the DBE. Once the DBE is up and running, the web sites, mailing lists, and discussion forums are a good start for information sharing and management, however, most SMEs need something more. Based on their experiences with current open source projects, the SMEs demanded the DBE to organize information in a more structured and clearer fashion. That could mean, for example, creating a suitable, focused ontology to ease the categorizing of information. Within ontology defined tags, web sites and discussion forums can be organized in a way that enables also novice SMEs to search for information autonomously. Easy access to information, openness in information sharing, and decision-making policies are ways of adducing the will of especially implementers and users towards DBE collaboration.

Now that all the sub-objectives and the main research question have been answered, it is time to assess the research, and the validity of its results. If this research could be started all over again, the most important thing to be performed differently would be the focusing of the subject. Now, the subject was quite extensive, and the aim of the study enabled quite comprehensive theoretical background studies. Additionally, including the lifecycle of the requirements in the research brought along a software engineering point of view, which broadened the background even more. It was quite a challenge trying to keep in boundaries of requirements engineering and not to write more about software engineering. A more focused research could have concentrated, for example, on requirements modeling practices in the SMEs, not the whole lifecycle of requirements management. On the other hand, when working for an ambitious project like the DBE that is aiming for challenging results, it is quite understandable that in the beginning of the project, the tasks and their contents still need some further clarifying. The fact that the DBE concept was still crystallizing when starting this research may have partially caused the challenges in focusing the objectives of this study. However, discussing so wide range of subjects is strength of this study. Additionally, the field of requirements engineering in open source projects had not been studied much, not to mention the requirements engineering practices between open source and proprietary software. Therefore, it was on the other hand fortunate that this study possessed a wide perspective on the subject.

The interview questions of the study could have been more precise as well. Although the questions were set as appropriate as possible at the time, only when analyzing the interview results it was discovered that the questions could have been more accurate. On one hand, the large scope of the study could be the reason for the generality of the questions. On the other hand, the need of focusing the questions was found during analysis of interview results, in other words, when the research problem started to become clearer to the researcher. Nevertheless, if the questions were set again, more “what” than “how” questions would be set. Additionally, the questions would be separated more clearly, since it seemed that in two-piece questions the latter questions often gained less attention in answers. Moreover, especially in questions concerning requirements engineering, more questions with alternatives would be set in order for the answers to be more comparable with each other.

Theoretical background of the study supported the understanding of the requirements engineering collaboration between proprietary and open source software very well. Especially the open source community characteristics helped to understand why requirements engineering is performed as it is in the different types of open source projects. A semi-structured approach was applied in the interviews allowing the interviewees to state their opinions quite freely. That brought more depth into the interview results. The chosen interviewees possessed quite the same kind of background, thus the results of cross-case analysis can be described quite reliable.

The aim to add to the understanding of requirements engineering process was reached. When it comes to what information flow and information management practices are needed when the software SMEs are collaborating with the DBE, the results of this study were not revolutionary. DBE collaboration of companies was classified in three groups according to the level of their collaboration capabilities and collaboration attitudes. However, another study in the DBE project at the same time had found there are four different types of companies regarding DBE collaboration. The other study was based on willingness of co-operation, in addition of co-operation abilities. It was satisfactory though that the results of the two studies were quite consistent. The engagement of the SMEs with the DBE should happen in different phases, as the information needs are different in different types of SMEs. However, this study found out that the requirements engineering practices of the SMEs might need some improving before the SMEs can effectively collaborate with the DBE. The result was exactly the opposite that was expected according to background knowledge. Additionally, this study brought out the needs and requirements of the SMEs for the DBE collaboration in general. Possibility of participating in decision-making was the requirement that came up most frequently; open and two-way communication was often mentioned as well. Moreover, the results of this study

are useful outside of the DBE. The thesis clarified the current state of requirements engineering practices in the SMEs in Tampere. Moreover, the use and knowledge of open source software was examined, as was attitude towards open source software as well. Since advancing both the requirements engineering practices and the knowledge and use of open source software of SMEs are focuses of TTC, the employer of the thesis, the results can be utilized in their future development plans and collaboration projects.

The case method that was used in this study has a known weakness; the results are not easily generalized (Aaltio-Marjosola 2001), which concerns this research as well. The sample of this study was quite small, only seven case companies, thus, the results cannot be generalized without further research. Only in the context of very similar companies that were researched, the results could be applicable, since the results are impressionable to the manners and practices of the case companies. Therefore, conducting the research with different organizations might cause different results.

It is easy to discover a few suggestions for further research. This thesis suggested ways of categorizing open source software collaboration of the SMEs. The categorization of open source collaboration according to willingness and ability would be an interesting subject to study further. Are the results of this research widely applicable and what are the differences between SMEs and bigger companies concerning open source software use and knowledge are questions that could be of interest at least to companies providing open source services, for example education.

Finding information of open source software requirements engineering practices was very difficult. Comprehensive study of requirements engineering practices in open source projects, categorized for example according to open source project size, would be quite interesting. Additionally, further research on requirements engineering collaboration would be interesting, as this study composed only an informal recommendation of it. The next step could be defining a requirements engineering process for the collaboration of open source projects and proprietary companies.

Some companies already collaborate with open source projects not only in requirements engineering but in the entire software engineering process as well. Thus, it would be interesting to study what the process is like. In that process basically, no money is transferred, therefore, it would be interesting to study what is the value that both parties in the process are given. The DBE project also evokes new research subjects. For example, research on the new revenue-generation models that the DBE enables would be quite useful not only to the DBE project in marketing the project, but to the actual European SMEs as

well. The number of further research subjects is huge, as is the number of fields of science that open source software can be connected to. Academic research on open source software offers potential both scientifically and commercially, as the open source community often seems to obey different practices, thus, offer new extensive possibilities.

References

Anon. 2004. Digital Business Ecosystem. Workpackage 28, DBE Training. Deliverable 28.1. DBE preliminary training resource plans and needs. Version 2.3. PMBE approval Brussels 2.7.2004. 122 p. (Unpublished, Project Internal Document)

Anon. 2003. Sixth Framework Programme, Priority 2.3.1.9. Networked Business and Governments. Contract for Integrated Project Annex I - “Description of Work”. Project full title: Digital Business Ecosystem. DBE Executive PM. 220 p. (Unpublished, Project Internal Document)

Awad, E. M. & Ghaziri H. M. 2004. Knowledge Management. New Jersey. Pearson Education. 456 p.

Beer, S. 1974. Designing Freedom. Toronto, CBC Publications.

Burckhard, H. & Smith, B. (eds). 1992. Handbook of Metaphysics and Ontology, 2 Vols., Philosophia Verlag, Munich-Wien.

Choo, C. W. 1998. The Knowing Organization How Organizations Use Information to Construct Meaning, Create Knowledge, and Make Decisions. New York, Oxford Press. 298 p.

Clippinger, J. H. III. (ed.). 1999. The Biology of Business Decoding the Natural Laws of Enterprise. San Francisco, Jossey-Bass Publishers. pp. 1-33.

Cocchiarella, N. 1972. Properties as Individuals in Formal Ontology, Nous 6, pp. 165-187.

Cocchiarella, N. 1974. Formal Ontology and the Foundations of Mathematics, in Nackhnikian 1974, pp. 29-46.

Cocchiarella, N. 1986. Logical Investigations of Predication Theory and the Problem of Universals, Bibliopolis, Napoli.

Cocchiarella, N. 1991. Ontology II: Formal Ontology, in Burckhard & Smith 1992.

Davenport, T. & Prusak, L. 2000. Working Knowledge. Boston, MA. Harvard Business School Press. pp. 1 – 24.

DiBona, C., Ockman, S. & Stone, M. (eds.). 1999. Open Sources Voices from the Open Source Revolution. Sebastopol (CA), O’Reilly & Associates. 272 p.

Dini, P., Kuusisto, T., Corallo, A., Ferronato, P. & Rathbone N. 2003. Toward a Semantically Rich Business Modeling Language for the Automatic Composition of Web Services. Proc. of eBusiness Research Forum 2003. Tampere, Finland, 2004. 20 p.

Dixon, N. M. 2000. Common Knowledge. Boston (MA). Harvard Business School Press. pp.12-13, 17-32.

Dorfman, M. & Thayer, R. 1997. Software Engineering. Los Alamitos (CA). IEEE Computer Society Press. pp. 80.

Fielding, R.T. 1999. Shared Leadership in the Apache Project. Communications ACM, 42, (4), pp. 42-43.

Fink, M. 2003. The Business and Economics of Linux and Open Source. Upper Saddle River (NJ). Prentice Hall. 242 p.

Genesereth, M. R. & Nilsson, L. 1987. Logical Foundation of Artificial Intelligence. Los Altos (CA). Morgan Kaufmann.

Goguen, J.A. 1996. Formality and Informality in Requirements Engineering (Keynote Address). Proc. 4th. Intern. Conf. Requirements Engineering. IEEE Computer Society. pp. 102-108.

Grady, R. 1992. Practical Software Metrics for Project Management and Process Improvement. Englewood Cliffs (NJ). Prentice Hall. p. 32.

Grant, R. M. 1996. Prospering in Dynamically-competitive Environments: Organizational Capability as Knowledge Integration in Organization Science, Vol. 7, No. 4, July-August 1996, pp. 375 – 387.

Haikala, I. and Märijärvi, J. 2000. Ohjelmistotuotanto. 7th edition. Helsinki, Satku . Kauppakaari. 414 p. (In Finnish)

Hansen, M., Köhntopp, K. and Pfitzmann, A. 2002. The open source approach - opportunities and limitations with respect to security and privacy. Computer & Security. Vol. 21(5), pp. 461-471.

Heikinheimo, H. 2003a. Open Source Software. Lecture of Software Business - Course. Institute of Business Information Management. Tampere University of Technology. 26.11.2003.

Heikinheimo, H. 2003b. Open Source Software Services. Master's Thesis. Tampere, Tampere University of Technology, Department of Industrial Engineering and Management. 140 p.

Hirsjärvi, S. & Hurme, H. 1988. Teemahaastattelu. 4th edition. Helsinki, Yliopistopaino. 144 p. (In Finnish)

Holland, J. 1995. The Hidden Order How Adaptation Builds Complexity. Addison-Wesley Publishing Company. 185 p.

Jacobson, I., Booch, G. & Rumbaugh, J. 1999. The Unified Software Development Process. Addison-Wesley. 463 p.

Kasanen, E., Lukka, K. & Siitonen, A. 1991. Konstruktiivinen tutkimusote liiketaloustieteessä. Liiketaloudellinen aikakauskirja 3. ss. 301 – 329. (In Finnish)

Kauffman, S. A. 1993. The Origin of the Order: Self-Organization and Selection in Evolution. New York. Oxford University Press.

Kruchten, P. 2000. The Rational Unified Process: An Introduction. Second Edition. Addison-Wesley. 298 p.

Kuusisto, T. 2004. Master's Thesis Meeting at Tampere University of Technology on the 17th of March.

Lerner, J. and Tirole, J. 2001. The open source movement: Key research questions. European Economic Review. Vol. 45(4-6), pp. 819-826.

Maier, R. 2002. Knowledge Management Systems Information and Communication Technologies for Knowledge Management. Heidelberg, Springer. 574 p.

Messerschmitt D. G. & Szyperski, C. 2003. Software Ecosystem Understanding an Indispensable Technology and Industry. Massachusetts Institute of Technology. The MIT Press. 424 p.

Moore, J.F. 1996. The Death of Competition. Leadership & Strategy in the Age of Business Ecosystems. HarperBusiness. 297 p.

Neilimo, K. and Näsi, J. 1980. Nomoteettinen tutkimusote ja suomalaisten yritysten taloustiede . Tutkimus positivismin soveltamisesta. Tampere, Tampereen yliopiston Yrityksen taloustieteen ja yksityisoikeuden laitoksen julkaisuja, Sarja A2 Tutkielmia ja raportteja 12. 82 p. (In Finnish)

Olkkonen, T. 1994. Johdatus teollisuustalouden tutkimustyöhön. Otaniemi, Helsinki University of Technology, Industrial Management and work psychology. 143 p. (In Finnish)

Pavlicek, R.2000. Embracing Insanity: Open Source Software Development. Indianapolis (IN). SAMS Publishing.

Prahalad, C. K. & Hamel, G. 1990. The Core Competence of the Corporation, in Harvard Business Review, Vol. 68, No. 5-6, 1990, pp. 79 – 91.

Rathbone, N. 2004. Evaluation of the DBE Codes. Email to Jaana Heliö on the 18th of June 2004.

Rich, R.F. 1981a. Knowledge in Society, in Rich, R. F. (ed.). 1981. The Knowledge Cycle. Beverly Hills (CA), USA. Pp. 11 – 39.

Rich, R. F. (ed.). 1981b. The Knowledge Cycle. Beverly Hills (CA), USA.

Sober, E. 1984. The Nature of Selection: Evolutionary Theory in Philosophical Focus. Chicago. University of Chicago Press.

Taylor, R.S. 1991. Information Use Environments, in Dervin, B. & Voigt, M. J. (eds.) Progress in Communication Science, 217-54. Norwood (NJ). Alex Publishing.

Taylor, R.S. 1986. Value-Added Processes in Information Systems. Norwood (NJ). Alex Publishing.

Weick, K. E. 1995. Sensemaking in Organizations. Thousand Oaks (CA), USA.

Yamaguchi, Y., Yokozawa, M., Shinohara, T. & Ishida, T. 2000. Collaboration with Lean Media: How Open-Source Software Succeeds. Proceedings of the Conference on Computer Supported Cooperative Work, CSCW. Philadelphia (PA). ACM Press. pp. 329-338,.

Yin R. K. 1989. Case Study Research: Design and Methods. Beverley Hills (CA). Sage.

Internet sources:

Aaltio-Marjosola, I. 2001. Case Study as a Methodological Approach. Metodix. Methods. Url: http://www.metodix.com/showres.dll/en/enindex (Accessed 6.4.2004)

Acharya, J. 2001. What is Knowledge. Pp. 2-5. Url: http://www.totalknowledgemanagement.com/ kmxchanges/whatisk.html (Accessed 25.2.2004)

Albertazzi, L. 1996. Formal and Material Ontology. Url: http://www.mitteleuropafoundation.it/Papers/LA/Formal%20Material%20Ont.pdf (Accessed 25.2.2004)

Clark, C. J. 2000. The UN/CEFACT Unified Modeling Methodology An Overview. United Nations, Centre for facilitation of Practices and Procedures for Administration,

Commerce and Transport. Url:
http://www.unece.org/cefact/docum/download/00bp030.doc (Accessed 25.3.2004)

Daffara, C., González-Barahona, J. M., Humenberger, E., Koch, W., Lang, B., Laurie, B., Aigrain, P., Cabirol, L., & Lacroix, M. 2000. Free Software / Open Source: Information Society Opportunities for Europe? Version 1.2. Open Source Software Licences. Url: http://eu.conecta.it/paper/Open_source_software_licenc.html#foot264 (Accessed 12.2.2004)

Dini, P., Nicolai, A. 2003. D.B.E. – The Digital Business Ecosystem. 14 p. Url: http://www.nachira.net/de/docs/dbe_summary_cc.pdf (Accessed 25.2.2004)

Free Software Foundation. 2003a. Copyleft: Pragmatic Idealism. Boston, Free Software Foundation Inc. Url: http://www.gnu.org/philosophy/pragmatic.html (Accessed 12.2.2004)

Free Software Foundation. 2003b. The Free Software Definition. Boston, Free Software Foundation Inc. Url: http://www.gnu.org/philosophy/free-sw.html (Accessed 12.2.2004)

Free Software Foundation. 2001a. Frequently Asked Questions about the GNU GPL. Boston, Free Software Foundation Inc. Url: http://www.gnu.org/licenses/gpl-faq.html (Accessed 12.2.2004)

Free Software Foundation. 2001b. What Is Copyleft? Boston, Free Software Foundation Inc. Url: http://www.gnu.org/copyleft/copyleft.html#WhatIsCopyleft (Accessed 12.2.2004)

Free Software Foundation. 2001c. Why ``Free Software" is better than ``Open Source". Boston, Free Software Foundation Inc. Url: http://www.gnu.org/philosophy/free-software-for-freedom.html (Accessed 12.2.2004)

Free Software Foundation. 1999. The GNU Lesser General Public License (LGPL), Version 2.1, February 1999. Boston, Free Software Foundation Inc.
Url: http://www.gnu.org/licenses/lgpl.txt (Accessed 12.2.2004)

Free Software Foundation. 1991. The GNU General Public License (GPL), Version 2, June 1991. Boston, Free Software Foundation Inc.
Url: http://www.gnu.org/licenses/gpl.txt (Accessed 12.2.2004)

Gruber, T. R. 1993. A translation approach to portable ontologies. Academic Press. Knowledge Acquisition, 5(2):199-220, 1993. Url:
ftp://ftp.ksl.stanford.edu/pub/KSL_Reports/KSL-92-71.ps.gz (Accessed 25.2.2004)

Guarino, N. 1998. Formal Ontology and Information Systems. In Guarino N. (ed.). Formal Ontology in Information Systems. Proceedings of the 1st International

Conference, Trento, Italy, 6-8 June 1998. IOS Press, 1998 p. 4 Url: http://www.loa-cnr.it/Papers/FOIS98.pdf (Accessed 25.2.2004)

Guarino, N. & Giaretta, P. 1995. Ontologies and Knowledge Bases. Towards a terminological clarification. Url: http://www.loa-cnr.it/Papers/KBKS95.pdf (Accessed 25.2.2004)

Hoskins, W. 1999. To All Licensees, Distributors of Any Version of BSD. University of California, Berkeley. Url: ftp://ftp.cs.berkeley.edu/pub/4bsd/README.Impt.License.Change (Accessed 20.2.2004)

Howe, D. 1997. “Ontology”. Free On-line Dictionary of Computing. Url: http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?query=Ontology&action=Search (Accessed 26.2.2004)

International Institute of Infonomics. 2002. FLOSS Final Report. University of Maastricht, Netherlands. Berlecon Research GmbH. Berlin, Germany. Url: http://www.infonomics.nl/FLOSS/report/ (Accessed 11.4.2004)

Merriam-Webster Online Dictionary. 2004. “Process”. Merriam-Webster, Incorporated. Url: http://www.m-w.com/cgi-bin/dictionary?book=Dictionary&va=process&x=14&y=11 (Accessed 27.2.2004)

Miller, J. & Mukerji, J. (eds.). 2003. MDA Guide Version 1.0. Object Management Group, Inc. Url: http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf (Accessed 23.3.2004)

Miller, J. & Mukerji, J. (eds.). 2001. Model Driven Architecture (MDA). Document number ormsc/2001-07-01. Architecture Board ORMSC. Url: http://www.omg.org/docs/ormsc/01-07-01.pdf (Accessed 23.3.2004)

Nachira, F., Chiozza, E., Ihonen, H., Manzoni, M., Cunningham, F. 2002. Toward a Network of Digital Business Ecosystems Fostering the Local Development. European Commission DG INFSO. Url: http://www.nachira.net/de/docs/discussionpaper.pdf (Accessed 25.2.2004). 23 p.

Nuseibeh, R. & Easterbrook, S. 2000. Requirements Engineering: A Roadmap in Finkelstein, A. (ed.). The Future of Software Engineering. Url: http://www.softwaresystems.org/future.html (Accessed 12.4.2004)

OMG. 2004a. OMG Background Information. Url: http://www.omg.org/news/about/ (Accessed 13.7.2004)

OMG. 2004b. Business Processes and the OMG, an Overview. Object Management Group. Url: http://www.omg.org/bp-corner/bp-files/The_OMG_BP_Corner_INTRO_Paper_3-2-04.pdf (Accessed 8.7.2004)

OMG. 2003. Meta Object Facility (MOF) Specification. Object Management Group. Url: http://www.omg.org/docs/formal/00-04-03.pdf (Accessed 24.3.2004)

OSI. 2004a. The BSD license. The Open Source Initiative.
Url: http://www.opensource.org/licenses/bsd-license.php (Accessed 20.2.2004)

OSI 2004b. The MIT License. The Open Source Initiative.
Url: http://www.opensource.org/licenses/mit-license.php (Accessed 20.2.2004)

OSI. 2004c. The Open Source Definition, Version 1.9. The Open Source Initiative.
Url: http://www.opensource.org/docs/definition.php (Accessed 3.2.2004)

Parviainen, P. (ed.). 2004. UKK – Usein Kysytyt Kysymykset, Avoin lähdekoodi. Url: http://www.opensourcefinland.org/fi/docs/ukk/ukk.html (Accessed 9.2.2004). (In Finnish)

Raymond, E.S. 1998. The Cathedral and Bazaar. Version 1.33. First Monday. 24 p.
Url: http://www.firstmonday.org/issues/issue3_3/raymond/ (Last modified 16.2.1998)

Scacchi, W. 2001. Understanding the Requirements for Developing Open Source Software Systems. IEE Proceedings – Software, Paper number 29840. Url: http://opensource.mit.edu/papers/Scacchi.pdf (Accessed 11.4.2004)

Siegel, J. & OMG Staff Strategy Group. 2001. Developing in OMG's Model-Driven Architecture. Object Management Group White Paper. Revision 2.6. Url: ftp://ftp.omg.org/pub/docs/omg/01-12-01.pdf (Accessed 23.3.2004)

SourceForge™.net. 2004. SourceForge™.net Home. Open Source Development Network. Url: http://sourceforge.net/ (Accessed 3.4.2004)

Spinellis, D. & Szyperski, C. 2004. How is Open Source Affecting Software Development. IEEE Computer Society. IEEE Software. Url: http://csdl.computer.org/comp/mags/so/2004/01/s1028.pdf (Accessed 5.4.2004)

W3C 2004. RDF Primer. W3C Recommendation 10 February 2004. Url: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/ (Accessed 18.5.2004)

Webster's Third New International Dictionary, Unabridged. 2002. "ontology." Merriam-Webster. Url: http://unabridged.merriam-webster.com (Accessed 26.7.2004).

APPENDIX 1 QUESTIONS FOR THE INTERVIEWS AT COMPANIES

Working Profile of the Interviewee

1. In which company you are working in?
2. How does the company generate or add value? (Classification!)
3. What stage of growth/maturity is the company at? (Regarding itself)
4. What ownership structure does the company have?
5. What management structure does the company have?

6. What is your role in the organization?
7. In what way is your work related to requirements engineering?
8. In what way is your work related to open source software?

Information needed by the SMEs

1. How do you elicit requirements of your software (Business Modeling, other)? From whom?
2. How are you requirements analyzed, designed (UML, RUP, MDA, MOF, other) and documented (e. g. locally, Web-based)?
3. How are requirements chosen for implementation (decision-making, prioritization)?
4. How do you get feedback of the software use and how is the feedback transformed into requirements?

5. Have you had any misunderstandings of terms or concepts with you customer and if you have, what kind?
6. Have you defined the terms of your customer domain? Would you consider that useful and how?

Co-operation with the Open Source Community

7. Have you had experiences of co-operation with an Open Source Community, if yes, what kind?

8. What would you expect of developing software together with an Open Source Community?

Ideas of the Requirements Engineering Recommendation

Finally, assess the characteristics for the communal requirements engineering recommendation of proprietary company and open source software project in the list below. They are ideas of what characteristics the co-operation could consist of.

1. Mark the characteristics that you agree to be the current characteristics of open source development. Comment why or why not if you like.
2. Mark the characteristics that are acceptable for you when co-operating with an open source project. Comment why or why not if you like.
3. Add some necessary characteristics/improvements of co-operation if they are missing in the list.

The characteristics of open source software development / collaboration:

1. Open source project communicates through Web-based forum(s) (discussion forum, mailing list, and newsgroup).
2. A (group of) maintainer(s) makes the decisions of the open source project. This means a proprietary company does not automatically have power over the project.
3. Power over an open source project is achieved gradually, by active working with the project (e.g. making a lot of quality code). For SME, this means assigning employee(s) to work in the project and/or contributing the community actively as a company (e.g. like IBM).
4. An open source project is released for production when the maintainers think its ready, not because of a timeline or a need of the SME.

5. Business Modeling is not performed in an open source project. SME may perform it itself and document it in the Web-based archives of the project.
6. The requirements in an open source project are informally discussed and documented in Web-based archives. If SME needs formal modeling, it may perform it itself and document it in the Web archives.
7. Analysis and design of the requirements in an open source project are performed and documented informally Web-based archives. If SME needs formal modeling and documentation, it may perform and document them itself in the Web archives.
8. If SME wants to participate in the decision making of the project, it has to participate actively also in the implementation phase, since

contributing quality code is what counts the most in gaining respect and finally power in an open source project.