NTNU
Innovation and Creativity

Selection of Open Source Components - A Qualitative Survey in Norwegian IT Industry

Marina Marinela Gerea

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Reidar Conradi, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

Open-source-software (OSS) becomes more and more used in software products developed by the industry. The motivation for this can e.g. be to reduce development/maintenance costs and to shorten time-to-market. There are several technical methods for selecting and evaluating OTS (Off-the-Shelf) components, but research shows that these methods are rarely used. OTS components come in two types: COTS (Commercial-Off-The-Shelf) and OSS (Open-Source-Software); this report focuses on OSS. The amount of systems with open source components is large and there are many questions to be asked such as: Where and how to find OSS components? How to evaluate them? How to learn the components? How to take care of knowledge about the chosen component? How to make best use of OSS? What component's version should be chosen and what could be done with respect to the maintenance and the new versions of the component? Without an effective way to select and evaluate OSS components, the time spent choosing the correct components may offset the advantages of using them.
This report presents the current state-of-the-art, a research design, and then presents the results of a survey performed in collaboration with Norwegian IT companies. The survey is based on the results of an explorative survey performed in the autumn of 2006; the interview guide was improved and we performed a larger survey. The state-of-the-art focuses on investigating existing research related to software development processes for component based-systems and OTS selection processes with the goal of finding research questions for research interviews. The results of the interviews may help software companies to understand the existing practices and improve their own processes.
This thesis presents two exploratory research goals and seven research questions. The research goals are: 1) to explore the company role: integrator of open source components, and 2) to explore their development processes with focus on OSS selection processes.
The contributions of this work include a literature study, and new empirical knowledge about OSS selection. This knowledge refers to the qualitative answers to the research questions. We have performed a qualitative descriptive study.

Assignment given: 10. January 2007
Supervisor: Reidar Conradi, IDI

TDT4900 System Engineering, Master Thesis
Spring 2007

Selection of Open Source Components – A Qualitative Survey in Norwegian IT Industry

Marina Marinela Gerea

Supervisors: Carl-Fredrik Sørensen and Reidar Conradi

Department of Computer and Information Science
NTNU Norwegian University of Science and Technology

Abstract

Open-source-software (OSS) becomes more and more used in software products developed by the industry. The motivation for this can e.g. be to reduce development/maintenance costs and to shorten time-to-market. There are several technical methods for selecting and evaluating OTS (Off-the-Shelf) components, but research shows that these methods are rarely used. OTS components come in two types: COTS (Commercial-Off-The-Shelf) and OSS (Open-Source-Software); this report focuses on OSS. The amount of systems with open source components is large and there are many questions to be asked such as: Where and how to find OSS components? How to evaluate them? How to learn the components? How to take care of knowledge about the chosen component? How to make best use of OSS? What component's version should be chosen and what could be done with respect to the maintenance and the new versions of the component? Without an effective way to select and evaluate OSS components, the time spent choosing the correct components may offset the advantages of using them.

This report presents the current state-of-the-art, a research design, and then presents the results of a survey performed in collaboration with Norwegian IT companies. The survey is based on the results of an explorative survey performed in the autumn of 2006; the interview guide was improved and we performed a larger survey. The state-of-the-art focuses on investigating existing research related to software development processes for component based-systems and OTS selection processes with the goal of finding research questions for research interviews. The results of the interviews may help software companies to understand the existing practices and improve their own processes.

This thesis presents two exploratory research goals and seven research questions. The research goals are: 1) to explore the company role: integrator of open source components, and 2) to explore their development processes with focus on OSS selection processes.

The contributions of this work include a literature study, and new empirical knowledge about OSS selection. This knowledge refers to the qualitative answers to the research questions. We have performed a qualitative descriptive study.

Keywords: Open Source, Open Source Selection, Structured Interviews, Survey.

Preface

This report has been written during spring 2007, as part of the course TDT4900 Software engineering, master thesis at the Norwegian University of Science and Technology (NTNU).

The context of the work is within the European ITEA project, COSI (Co-development using inner & Open source in Software Intensive products). The Norwegian COSI wants to enable the Norwegian IT sector to fully exploit the benefits and advantages of open source software (OSS).

This work consists of a literature study and a survey about development processes with OSS components in Norwegian IT companies. It is particularly of interest to discover the actual OSS selection processes in the industry.

The work was performed under supervision of Dr. Carl-Fredrik Sørensen and Professor Reidar Conradi. My supervisor, Dr. Carl-Fredrik Sørensen, helped me very much by providing most of the IT companies names in Norway and then later in getting in touch with them. I want to thank both Dr. Carl-Fredrik Sørensen and Professor Reidar Conradi for providing insightful input and valuable feedback all the time. The group of people working on the COSI project at IDI, NTNU was very enthusiastic and eager to provide feedback any time I needed it. I was glad to be part of the group, even for a short period of time, and I am grateful to them. I would like to thank to Øyvind Hauge from NTNU and Claudia Ayala from The Technical University of Catalunya who helped me and encouraged all the time.

I would also like to thank my former colleagues at SINTEF Marine Environmental Technology for their support and encouragement, especially to Dr. Mark Reed and Dr. Tore Aunaas.

Last by not least I would like to thank my husband Constantin Vili, my daughter Carolina Gabriela and my family for their help and support, they gave me the power to overcome the difficult periods.

We have performed nine interviews during the spring 2007. We would like to thank all the respondents who took time to answer the interview: Frank Forseth from Visma, Emil Urnes from Sirius IT, Christer Grimsæth from TietoEnator, Morten Nørsvold from WebOn, Eivind Tagseth from Abeo, Trond Lindanger from DKDigital, Jo Arve Aamaas from Commitment, Robin Smitsrød from Grieg Multimedia and Audun Kvasbø from Riventi.

Trondheim, June, 2007

_____

Marinela Gerea

Contents

List of tables

List of figures

Part 1 - Introduction

1. Introduction

1.1 Problem Outline

More and more companies use OTS (Off-the-Shelf) components in their daily software development. The OTS components can be distinguished into two types: COTS (Commercial-Off-the-Shelf) and OSS (Open-Source-Software). As described in [H.K.N.Leung 2003], "use of commercial-off-the-shelf (COTS) products is becoming an acceptable software development method. With the increased number of available COTS components, the time spent on choosing the appropriate COTS products could easily offset the advantages of using them". Therefore, it is very important to effectively select COTS components.

Our contributions are: a literature review, a research design, a survey, all these with the goal of discovering better methods to select and evaluate OSS components that can help companies in their development processes. It is necessary to empirically investigate how software companies select components, in different projects and different application domains.

This research is mainly based on our results from the interviews performed in the depth study of autumn 2006.

The contributions from the depth study of autumn 2006 are: a literature study which provides descriptive information about the state of the art, the qualitative answers to the research questions, a platform for future work mainly for us but also for others who have similar interest, and an interview guide.

These results are presented as sixteen descriptive findings (DF) (see Table 1).

DF1: Usually it is the software architect who initiates the work related to harvesting. The work of harvesting is usually done by the software developer and the final decision about integrating an OTS component is usually done by project manager or software architect.
DF2: The license price is the most important business issue when selecting and evaluating OSS components, but also the maintenance cost is quite important.
DF3: The functionality is the most important technical issue when selecting and evaluating OSS components. When the functionality criterion is met, the quality of code and design, of architecture and documentation are also important. Support for standards and standard compliance play also an important role.
DF4: The availability of component for test and use is the most important organizational issue when selecting and evaluating OSS components.
DF5: The fact that people have experience with the OSS components or that the components are written in programming languages people know, are also important issues when selecting and evaluating OSS components.
DF6: Unsuitable license and bad code quality are the most important properties which makes a company to discard an OSS component.
DF7: The number of selection processes and the time used for selection is proportional with the complexity of the component to be integrated.

DF8: Companies do not use any formal processes for the selection of OTS components.
DF9: Companies find OTS components by searching on specialized sites or by searching using specialized search engines.
DF10: Components are tested before integration by making a small prototype.
DF11: It may happen to realize in the end of the development process that the selected OTS component is not suitable.
DF12: Companies consider the last or the last stable version when selecting and evaluating OSS components.
DF13: Companies update to a new version when additional functionally is needed or when bugs need to be corrected.
DF14: People documents the decision besides selecting and evaluating OTS components and this is usually done as part of the product/project documentation.
DF15: Small or medium size companies do not keep a knowledge repository about the selected OTS components, only the large companies do.
DF16: Usually there are people responsible for the OTS components in each company.

Table 1: Findings from the depth study.

1.2 The Purpose of this Thesis

The purpose of this thesis is described in terms of a project description, the overall design goal and several research questions.

The project description contains two clear tasks: to do a literature study, and to prepare a survey. The literature study focuses not only on purely academic papers, but also on papers containing aspects of the practices in the software industry. A special focus was on papers describing the selection processes of OSS and COTS. Even if our focus is on open source components, we investigated both types because they have things in common and it is interesting to see the difference.

Based of the results of the explorative survey performed in the autumn of 2006, we have improved the questionnaire and then performed a survey with more respondents, where the respondents have been drawn randomly from a larger population. This work is a qualitative descriptive study since we have interviewed only nine respondents. We are concerned with comparison across company size.

Research method: grounded theory

Grounded theory is one type of qualitative research, which is heavily relying on the data that is collected. It aims to use the data for building theoretical constructions unlike other methods more used in natural sciences where data is used to evaluate existing hypothesis. By data, we mean transcripts of interviews. The analysis can be word-by-word, sentence-by-sentence, or on a more abstract level to find concepts. Another much used technique is to apply “axial coding” – to place concepts along an axis [Strauss and Corbin1998].

Research goals:

1. Explore the company role: integrator of open source components.

2. Explore their development processes with focus on selection processes of OSS components.

We want to discover what the selection processes are when companies integrate OSS components into a system or application. We are not referring to infrastructure OSS (e.g. Linux) or open source development tools (e.g. Eclipse, MySQL).
The object of study is the process of selection of open source components in industry.

Research questions:
RQ1: Who (which role in a company) initiates and performs the work related to OSS harvesting and when in the development process?

RQ1.1: Who (which role) initiates the OSS harvesting in a company?

RQ1.2: Who (which role) does the work related to the harvesting?

RQ1.3: Who (which role) takes the final decision about integrating an OSS component?

RQ1.4: When in the development process does one select OSS-components?

RQ2: What are the motivations for and experiences of using OSS components?
RQ3: What is the current process of selecting OSS components?
RQ4: How to find OSS components?
RQ5: What are the evaluation criteria when selecting OSS components?
RQ6: What versions are considered and how to deal with new versions?
RQ7: How to maintain the knowledge about the selection processes and the knowledge about the selected OSS components?

Research context:

The previous depth study was performed in the context of the ITEA2 COSI project. A short description of the Norwegian COSI project is presented below:

The Norwegian sub-project is sponsored by the Norwegian Research Council and led by ICT-Norway. In addition to ICT-Norway, there are also four industrial partners (Keymind, Linpro, ITFarm and eZ Systems), and one academic partner, NTNU.

The goal of the Norwegian COSI project is process improvement of development processes related to open and inner source software development. The Norwegian project is further responsible for deliverables to the European project related to development processes. Each industrial partner is responsible for reporting their processes related to open source, and NTNU is gathering these practices and presenting them to the European project.

This thesis has however been performed as separate research outside the scope of the COSI project.

1.3 Apparatus

We collected *a list of Norwegian companies* (IT companies and non-IT companies which have a software development unit) by different means in order to apply stratified random sampling for finding subjects for interviews.

We have used different ways to find these companies, their size and focus, and we ended-up with about 620 companies.

We used mostly Internet to find out the main focus and the size of the companies, whether a company is IT or not, and whether they do software development and integration of OSS components or not.

Then we divided the whole list into three sub-groups based on size and applied random sampling within the strata of the three groups of ICT companies. Due to difficulties to find the right person in non-ICT companies, we have only used the ICT companies. We have divided the IT companies in three groups because we are concerned with comparison across company size.

The goal was to have three companies in each group. We randomly selected companies from each of the three sub-groups and took contact with these companies. Some companies could either not answer or were outside the scope of this study. We ended up with three subjects in each group (totally 9 respondents). Because we only had 9 respondents for the interviews, our study is qualitative. The goal is to discover the practices about selection of OSS components in the different Norwegian IT companies. We believe that the sampling method has given us a better coverage of the practices within the companies, and thereby a better and broader representation than the convenience sample from the depth study.

A more detailed description of the population and sampling method of study will be given in the Part 3 of the thesis.

1.4 Contributions

The contributions of this work can be divided into two parts: the literature study and new empirical knowledge regarding the industrial practice about selection of OSS components.

The first contribution is the *literature study*. This study provides descriptive state-of-the-art information related to traditional models, OSS development process and software development processes for Component Based Systems (CBS) and OTS selection processes.

The second and most important contribution is *new empirical knowledge*. This knowledge refers to the qualitative and quantitative answers to the research questions. We have performed a qualitative descriptive study.

Having as input several hypotheses and observations from the depth study in 2006, we have improved our questionnaire and performed a larger survey. We have proposed descriptive findings based on our results, which are presented in Chapter 6. Some of the most relevant findings are presented below in Table 2:

Issue	Descriptive finding
When in the development process are the OSS components selected?	DF2: Large components are usually selected at an early stage during the development process, while small components can be selected anytime during the development process.
Motivations to integrate OSS components	DF3: Higher quality, shorter time to market, and cost are the principal motivations to integrate OSS components into a system or application.
Selection process	DF4: Companies do not use any formal processes for the

	selection of OSS components.
Evaluation criteria	DF6: Matching functionality and standard compliance are the most important technical issues when evaluating an OSS component for integration into a system or application.
	DF7: The vitality is the most important organizational issue of the OSS community when evaluating a component for integration into a system or application.
	DF9: Unmatched functionality, unsuitable license and the difficulty to integrate are the most important properties which may make a company to discard an OSS component.
Version issues OSS	DF11: Companies usually stay at the latest stable version and they only update to a newer version when the gain to update is bigger than the cost to update.
Integration of OSS components	DF13: In large companies, replacement of selected OSS companies at some point during the development process happen less often than in medium size companies and in medium size companies this happens less often than in small companies. When this happened, then it was usually done during the development phase and less often in the testing phase. Thus, company size has an effect of the component replacement rate.
Activities after the integration of OSS components	DF14: The large companies document the rationale behind the choice of the selected OSS component more than the medium companies. The small companies do not document their rationale.
	DF15: Most companies keep a knowledge repository with all the OSS components used in different projects; some companies keep also information about the selected OSS components and they reuse this information later.
	DF16: Most companies have a person who is responsible for the OSS components, even if this is not always a formal role.

Table 2: The most relevant findings of the thesis

Our work creates a *platform for future work* for anyone who has similar interests.

We have developed an *interview guide* and a *help for the interview guide* and performed several structured interviews to assure/improve it. The help for the interview guide can easily be changed to a questionnaire that can be used to study a larger sample and thus be able to get quantitative results with statistical relevance and thus improve the external validity of this work.

1.5 Report Structure

This report is structured into several chapters.

Chapter 1, *Introduction*, is this part of the document.

Chapter 2, 3 and 4 presents the *Pre-study*. They contain the results from the literature review and give an overview over the existing selection methods described in literature for Off-the-shelf (OTS) components.

Chapter 5 presents the *Research design*. It presents the project timeline, the research questions, the empirical strategies used, the interview design, and the research context.

Chapter 6, 7 and 8 present the *Results, Discussion, and Conclusions* of the study.

After the references, the last part contains several appendices like: Glossary, The Open Source Definition, a section about OSS Development Process, the Interview Guide and a Help for the Interview Guide.

An overview of the content of this report can be found in Table 3.

Part	Chapter	Content
Introduction	1	Introduction
Pre-study	2	An introduction to Open Source
	3	Traditional Models, OSS Development Process and Software Development Processes for Component Based Systems (CBS)
	4	OTS Selection processes
Research	5	Research agenda
Results, Discussion and Conclusions	6	Results
	7	Discussion
	8	Conclusions and future work
References	-	References
Appendices	A	Glossary
	B	The Open Source Definition
	C	OSS Development Process
	D	Interview Guide
	E	Help for the Interview Guide

Table 3 : The content of this report

Part 2 – Pre-study

2. An introduction to open source

This chapter aims to give the reader a short introduction to open source by presenting the definition of open source and COTS, followed by a short description of organizational issues and software development processes.

As indicated already, there are two types of OTS components: OSS and COTS (see below).

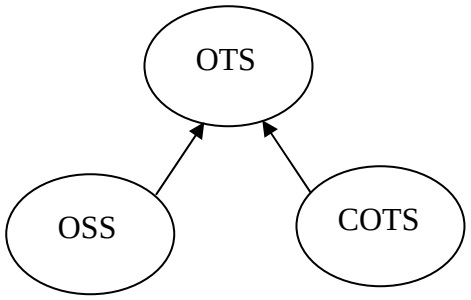

Figure 1: OTS, OSS and COTS components

2.1 Historical Background of Open Source

This section will give a brief history of OSS, with some of its most important moments.

2.1.1 The Early Days of Computers and Open Source

The computer history starts in the 1940s, when computers were primarily used for scientific problem solving [Feller and Fitzgerald2002]. The users of computers at the time were typically scientists with strong mathematical or engineering background who developed their own programs.

During the early 1950s, the use of computers began to spread beyond scientific problem solving to address the area of business data processing. Writing programs at that time required much creativity and resourcefulness of the programmer. It was recognized a major achievement to get a program to run at all. Thus, if software worked it was shared widely. In 1953, the Project for Advancement of Coding Techniques initiative was a collaboration between the military and aviation industry [Leonard2000]. This is an early formalized example of free sharing of software. Other examples of free sharing of source code at the time involved user groups at IBM and DEC. The openness of these and other similar communities was a significant step toward the foundation of the open source community we know today.

2.1.2 The Early Days of UNIX

During four weeks in the summer of 1969, Ken Thompson created the first version of an operating system called UNICS (Uniplex Information and Computing Services). Unics would later be renamed UNIX [Weber2004]. Certainly none of the early UNIX programmers could have foreseen that more than thirty years later, UNIX would remain

a mainstream operating system for researchers, custom applications, major business software, and perhaps most important the Internet.

2.1.3 UNIX and the Internet

In 1968, Pentagon Defence Advanced Research Project Agency (DARPA) started the Advanced Research Project Agency Network (ARPANET). DARPA chose Unix for linking together the ARPANET research nodes.

It was need for a communication protocol within the meta-network, thereby catalyzing the development of the [1]TCP/IP protocols (the rules of communication that currently underlie the Internet) [Weber2004]. The TCP/IP protocol was developed by the Berkeley Software Distribution (BSD) group, was widely disseminated, and contributed to the rapid diffusion of Internet. Other important Internet utilities that were developed as part of BSD were Sendmail and BIND.

2.1.4 The Free Software Foundation

One of the most significant milestones in the history of OSS was the establishment of the Free Software Foundation (FSF) by Richard Stallman in 1985. After resigning from MIT, Stallman devoted his attention to create a suite of free software products, the [2]GNU family. In the GNU manifesto, Stallman (1985) coined the term "Free software", thus formalizing a process that had been going on in a relatively ad hoc fashion in the past. The ambiguity of the word "free", having both the meaning "unfettered" and "gratis", lead to the eventual coining of the term "Open Source" later [Weber2004]. Stallman wanted to use existing copyright law to guarantee some basic rights to all future users of software. Free, as in freedom not as in gratis, was one of the most important issues. With this in mind, he formed the widely used General Public License (GPL). Under this license everyone should have permission to run, copy, modify, and redistribute software and the source code, but you can not add any restrictions or limits to this freedom.

2.1.5 Linux – A UNIX Kernel

Generally, some of the necessary technology already existed through GNU. What was missing for PCs was a UNIX kernel.

In 1991, Linus Torvalds wanted to create a UNIX-like operating system for the IBM PC 386 series. Torvalds succeeded in attracting a great deal of support worldwide. The Linux developer's community represents the largest collaborative project in the world history.

Ironically, Linux has become more popular than UNIX, the operating system on which it was initially based on and it is currently the most widely ported operating system available on the PC platform [Webwe2004].

---

[1] TCP/IP means Transmission Control Protocol/Internet Protocol. Internet is the inter-connection of all the networks in the world. The IP protocol allows the links between the different machines in this network.

[2] GNU is a computer operating system composed entirely of free software. Its name is a acronym for *GNUs not Unix*, which was chosen because its design is Unix-like, but differs from Unix by being free software and by not containing any Unix code. GNU was founded by Richard Stallman and was the original focus of the Free Software Foundation (FSF).

2.1.6 Apache – An Open Web Server

The next milestone in the history of OSS was the development of the Apache HTTP Server, begun in February 1995 by a group of volunteers. The Apache server was based on a series of patches to the web server initially developed by Rob McCool at the National Center for Supercomputing Applications (NCSA). The NCSA server had been popular and many individual webmasters had developed extensions and patches [Webwe2004].

The Apache server is, according to the Netcraft survey, the most widely deployed Web server at the time of this writing [Mockus et al.2002].

2.1.7 Mozilla – A Free Web Browser

One of the most important initiatives in the history of Open Source Software is the Mozilla Project; in addition to the software that it has produced, it had an enormous impact in promoting corporate and media awareness of the concept. In January 1998, Netscape announced that the source code for their browser would be made available, and Mozilla was the name chosen for the project. It was an extremely courageous step of making the source code of their browser available. A special pair of licenses, the Mozilla Public License (MPL) and the Netscape Public License (NPL), was created for the project of releasing the source code. Within hours of the source code being made available on March 1998, developers around the world were submitting patches to the project. By the end of 1998, Netscape was starting to regain market share.

2.2 Definition of Open Source

There are several views of what Free Software (FS), Open Source (OS), Open Source Software (OSS), Free Open Source Software (FOSS) or Free/Libre Open Source Software (FLOSS) is.

The Free Software Foundation (FSF) and it's pioneer Richard Stallman have their Free Software Definition. The definition refers to four kinds of freedom [Free Software Foundation2006]:

- The freedom to run the program, for any purpose (freedom 0);
- The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this;
- The freedom to redistribute copies so you can help your neighbor;
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this.

The Open Source Initiative (OSI) has on the other hand its Open Source Definition [opensource2006]. Bruce Perens [opensource2006] wrote a definition of open source as a pragmatic reaction to the misinterpretation of free software as "gratis software". The Open Source Definition is currently available in version 1.9, see Appendix B.

The definition contains the following 10 requirements to products which want to call themselves open source. The list below is partly adopted from Understanding Open Source Software by Mark H. Webbink [Webbink2003]:

- No royalty or other fee imposed upon redistribution;
- Availability of source code;

- Right to create modifications and derivative works;
- May require modified versions to be distributed as the original version plus patches;
- No discrimination against persons or groups;
- No discrimination against fields of endeavor;
- All rights granted must flow through to/with redistributed versions;
- The license must not be specific to a product;
- The license must not restrict other software;
- The license must be technology-neutral.

2.3 Definition of Component

Components are defined in various ways depending on the viewpoint. Several technical definitions exist. The definition of Heineman and Council [Heineman and Council2001] is: A software element that conforms to a component model, which can be independently deployed and can be composed without modification according to a composition standard.

The granularity of the component in practice/marketplace is generally the same or larger than the theoretical defined above. Components grouped as .NET, ActiveX/COM, Java, JavaScript, AJAX, C++/MFC, DLL, and VCL are all regarded as components in the industry and in the component marketplace [Componentsource2007].

Components can be built in-house, acquired from third-parties, or just be downloaded from the websites of the OSS communities.

In our study, we investigated the components with similar granularity as in industrial practice and component marketplace. However, we chose to exclude infrastructure OSS (e.g. Linux) and open source development tools (e.g Eclipse).

2.4 Definition of COTS

The terms OSS, COTS, and OTS are used everywhere in the report, thus, we need to provide definitions of these terms.

[Carney and Long2000] consider that the existing labels COTS and OTS are not sufficient, and many people do not understand exactly the meaning of COTS. If a component was obtained rather than made, then it is called Off-the-Shelf (OTS). A Commercial-off-the-shelf (COTS) component can be modifiable or not. The solution is to separate attributes along two axes*: source* and *modification*, and locate the component in a graphical space.

Placing a component on the *source* axis, the component can be:

- Independent commercial item acquired from a third party: This is a COTS
- Custom version of a commercial item: Has some degree of COTS properties;
- Component produced under a specific contract: Has some degree of COTS properties
- Existing component obtained from external sources (for example, a reuse repository): This is a previously developed component.
- Component produced in-house: This is a totally non-OTS component.

Placing a component on the *modification* axis, the component can have:

- Very little or no modification.
- Simple parameterization.
- Necessary tailoring or customization.
- Internal revision to accommodate special platform requirements:
  Include code revision to use special operating system protocols or real-time constraints.
- Extensive functional recording and reworking:
  A drastic modification, fundamental alterations such as removing safeguards or internal access control.

OSS components have always the source code open while COTS components may either have the source code open or closed (about 1/3 of COTS components are shipped with the source code available). Therefore, it is important to distinguish between a COTS component and an OSS component. Placing a component on the *source* axis, shows the distinction between a COTS and a non-COTS component (which can be OSS component or a component produced in house).

2.5 Organizational Issues

2.5.1 The Roles in OSS Community and Open Source Projects

People can take different roles in open source projects like: *joiner*, *newcomer* and *developer* [Krogh et al.2003]. A joiner is someone who is on the e-mail list but does not have access to the CVS repository. A newcomer is someone who has just begun to make changes in the CVS repository. A developer is someone who has moved beyond newcomer stage and is contributing code to the project.
Usually, a significant period of observation is necessary for a joiner to contribute to the technical discussion. A joiner lurk silently on the developer list and learns as much as possible before making any technical contribution, rather than entering into the developing list asking general questions.

There are different roles in an open source project, but people can often have several roles, e.g., both developer and user [Berquist and Ljungberg2001].
These roles as shown in Figure 2 are:

- The owner of an OSS: is the person (or the group) who started the project and has exclusive the right, recognized by the community at large, to redistribute modified versions of the software.
- Core developer: it is a group of core developers who write most of the source code, design, and take the important decisions in an open source project
- Developers: a wider group than the core developers who repairs defects.
- Problem reporters: this is even a larger group
- System testers
- User support: this task is performed mainly by some product users voluntarily providing answers to the questions of other users
- Users: it is a huge number of users; in addition to individuals, even large organizations and companies have become users. By using OSS, these actors are

important contributors. Commercial companies tied to open source projects provide additional resources for development and help to promote the open source packages and drive them into the mainstream.

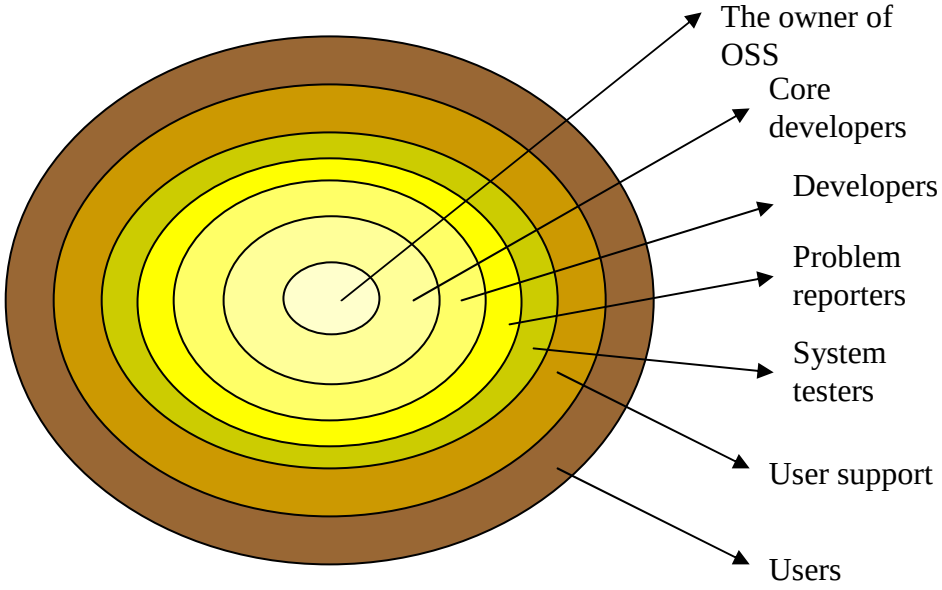

Figure 2 : The roles in open source projects

2.5.2 Open Source Licenses

Each open source project includes conventions that all participants are supposed to follow. Tacit and not written rules are widely accepted in the project community rather than other legal frameworks. Furthermore, licenses are the most important institution in the governance structure of open source projects [Bonaccorsi and Rossi2003].

A license is basically an agreement between the user and the developer on how software can be acquired and used. Unlike proprietary or commercial software, one of the hallmarks of OSS is that there are no unit or per-seat licenses – you can take one copy of the software, and install it on as many machines as you want, with no added license cost. It is not necessary to track licenses.

Opensource.org (www.opensource.org) lists 21 approved licenses, which are the standards licenses that the Open Source Initiative (OSI) has certified as valid. The base license determines how a developer can use an open source software component. From a commercial firm's perspective, the license must explicitly permit the distribution of software build from modified source code [Madanmohan and De2004].

The most used licenses are:

- GNU General Public License (GNU GPL)
  It is the earliest open source license, created in 1980s to distribute the GNU project software. Most open source software to this date has been distributed under GPL. It is a copyleft license. The main characteristic of the GPL copyleft license is its viral nature: a piece of a GPL code inserted in a given program forces that program to be

protected by the same license agreement; every program containing a piece of code released under the GPL, must be released under the GPL too. This license is therefore controversial.
A company can implement a component under the GPL license into a product and benefit from the component by using the final product internally. If, however, an organization creates a commercial software product using a component licensed under the GPL, and sell the software, the product itself, in its entirety, would be considered a derivative work as defined by the GPL.
A modified license, the Lesser GPL (LGPL) was created when this proved impractical. The LGPL differs from the GPL in two ways. Firstly, it is intended for use with software libraries. Secondly, the software may be linked with proprietary code, which is precluded by the GPL.

- Berkeley Software Distribution (BSD)
  This is also an early license. The company is not required to grant any particular license right. Its main requirement is the retention and acknowledgment of previous contributors' work.
  It is a free license. It is a copy-right license. A commercial company can take open source software under a BSD-style license and use it to create a proprietary product for which source code is not made available. Many commercial companies use this license in their business and software they develop, e.g.: Yahoo, Apple, Microsoft.

- The Artistic License
  The Artistic License was designed to be used specifically with Perl. The Artistic License is perhaps best thought of an attempt to create an open source license that eliminates or mitigates the more controversial aspects of the GPL. In particular the Artistic License differs from the GPL in the following ways (among others) [Hecker200]:
    - The Artistic License encourages users to make modifications freely and publicly available, but allows exemptions from such a requirement in the cases where the derived works are used only within an organization and are not publicly distributed.
    - The Artistic License allows the original work or derived works to be embedded "invisibly" in a proprietary program. Under the GPL the proprietary program would be considered a derived work also subject to the GPL.

- Mozilla Public License (MPL)
  The license is a mixture of BSD and GPL. Separate non MPL files may be licensed under other terms. Companies must state MPL terms to any file that contain MPL code.

- Apache License
  The Apache License (versions 1.0, 1.1, and 2.0) requires preservation of the copyright notice and disclaimer, but it is not a copyleft license - it allows use and distribution of the source code in both free/open source and proprietary/closed source software.

2.6 Summary

We started this chapter by presenting a brief history of OSS. It is important to observe the milestones and how the OSS evolved from its earliest days.
We continued by referring to the definition of OSS in Appendix B. A definition of component in general from the technical and marketplace point of view is also necessary. Many times, people are confused about the difference between OSS, COTS, or in-house components, mainly because they all may have the source code opened. Therefore, a definition of COTS and how it is different from OSS and in-house component was necessary.
Regarding the organizational issues paragraph, it is important to see the different roles in open source projects and the most important institution in the governance structure of open source projects (i.e. licenses).
Our research goal is to explore the company role: integrator of open source components. This company role belongs to the “users” role in open source projects.
The open source licenses part gives an overview over the most used licenses. The licenses are always considered when people select and evaluate open source components.

3. Traditional Models, OSS development process and Software Development Processes for Component Based Systems (CBS)

We start this chapter by presenting the traditional and the composed models usually used when developing proprietary software, followed by development processes used when developing OSS. Very often, proprietary software consists of COTS.
There are two types of component-based systems: OSS-based systems and COTS-based systems. The development processes for OSS-based systems and the development processes for COTS-based systems have similarities and differences. However, both processes are somewhat different from the traditional and the composed process models, where no components are selected and integrated into the final system or application.

3.1 The Traditional and the Composed Process Models

Before reviewing CBS development processes, it is important to list the so-called traditional and composed process models.

The *traditional* process models are:
- **Code and Fix**: Developers begin work without a set of good requirements or a vision, writing code as long as there is time and money. This approach has no planning and therefore yields early results. High maintenance and rework effort are the result of this model. The code developed with this method has usually poor quality.
- **Waterfall**: This model is a sequential, document-driven methodology. To advance from the current phase to the next phase, the project team must review and release an artifact in the form of a document. [Royce87]
- **V-model**: This model is an extended waterfall model, adding more details on the validation and verification side. For each phase in development, there is an associated phase to verify and validate the result produced. [German V-model]
- **Spiral**: The spiral model breaks the project down into risk-oriented subprojects [Boehm88]. The advantage of this model is the breakdown of development into small pieces, so that the areas of highest risk can be tackled at the beginning of the project.
- **Evolutionary Prototyping**: In this model, the system starts with an initial idea or proposal, which is then prototyped and released to the customer in incremental releases based on feedback.

The *composed* (incremental and iterative) process models, which cover several aggregations, are:
- **Unified Process (UP)**: UP is a risk- and user-driven, architecture-centric, iterative, and incremental software development model. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process or RUP.
- **Agile models:**
  - **Adaptive Software Development (ASD):** The ASD model is an iterative, risk – and mission- driven, component-based and change-tolerant process [Highsmith99]. It works well for small teams (four to eight people) when requirements are uncertain or domain knowledge is missing.
  - **Extreme Programming (XP)**: The XP model [Kent99] is a flexible, lightweight, people- and result-oriented development process that allows for requirements

changes at any time during development. XP defines 12 core practices that are, according to Beck, essential to the success of an XP project.

- **Scrum**. The characteristics of this process model are: tries to have everybody involved, the requirements from customers are prioritized, daily meetings with the customers [Wikipedia.org].
- **Crystal Clear:** Crystal Clear can be applied to teams of up to 6 or 8 colocated developers working on systems t.hat are not life-critical. The Crystal family of methodologies focuses on efficiency and habitablity as components of project safety. Crystal Clear focuses on people, not processes or artfifacts [Wikipedia.org]. The most important and required properties of Crystal Clear are:
    - Frequent delivery of usable code to users;
    - Reflective improvement;
    - Osmotic communication preferably by being co-located.

Agile methods are a response to the more rigorous and traditional approaches to software development which emphasize the (perceived) importance of predictive planning, the use of appropriate processes and tools, and the need for documentation. Agilists offer approaches which stress collaborative practices, face-to-face communication, collaboration with the customer and the importance of the individual and the team [Sharp and Robinson2004].

3.2 Software Development Processes for Component-Based System (CBS)

More and more companies are using components (OSS or COTS) when developing systems or applications.

Nowadays there is a significant amount of work that identifies issues or proposes frameworks for pursuing further research for improving new software development processes for component-based system (CBS).

Most of the research focuses on development processes using COTS. Even if our focus is on OSS, we also present both the COTS-based development process and the OSS-based development process to observe similarities but also differences.

Our research goal is to explore the role: integrator of OSS components and their development processes with focus on the selection process. Integrating an OSS component into a system or application is an important activity in an OSS-based development process. In order to understand the selection process of OSS components it is important to understand the context in which this activity occurs (i.e. OSS-based development process).

3.2.1 Software Development Processes using COTS

In [Brownsword et al.2000], the changes required to address CBS development are identified taking into account real-life lessons.

In a custom-development approach, the development team identifies requirements (technical and non-technical, other constraints such as cost and schedule, etc), defines an architecture, and then undertakes (custom) implementation. Instead, with COTS-based systems, developers must consider requirements, architecture, and marketplace

simultaneously. Any of the three might affect the other two, so none can proceed without knowledge and accommodation of the others. Furthermore, the activities performed for CBSs are cyclic: these trades-offs recur frequently throughout the system's lifetime. Sometimes the differences between CBS and custom-development processes are subtle. Often the differences are not in what is done but rather how or when or with what marketplace considerations the CBS activity occurs within.

[Morisio et al.2000] report about adopted COTS-based processes and propose a new one. New activities identified in COTS-based processes are: product evaluations, product familiarization, and vendor interaction (of technical, administrative and commercial nature).

[Li2006] shows that the development process used in the projects using COTS components is a customization of the traditional development process due to the use of COTS components. It is also indicated that there are some common new activities (e.g. make vs. acquire decision, COTS component selection, learn and understand the COTS components) and one new role (e.g. a COTS component knowledge keeper) is added in COTS-based development processes. The possible variations are when and how to perform these new activities.

3.2.2 Software Development Processes using OSS

In most cases, the process that builds applications on existing OSS is largely similar with the process of developing COTS-Based Applications (CBA). Selecting correct OSS or COTS, adjusting them as needed, and integrating them are essential actions during system development lifecycle [Huang and Yang2005]. However, there is a significant difference between OSS-Based Applications (OBAs) and CBA development. In building OBAs, developers not only need to integrate existing OSS, but also need to modify existing OSS because of their quality or their functionality. Therefore, developers can not use the process of developing CBA to build OBAs directly.
The authors describe a process for building OSS Based Applications (OBAs), it is a process with several activities:

- Identifying High-level Requirements & Candidate OSS;
- Designing High-level Architecture;
- Eliciting Requirements and Establishing Assessment Criteria;
- Assessing OSS according to criteria;
- Adjusting Criteria or High-level Architecture;
- Constructing In-house Products if suitable OSS do not exist;
- Improving and Integrating OSS;

3.3 Summary and Discussion

Before presenting the OTS selection methods, we have provided an overview over the traditional development process, the software development processes using COTS and OSS in order to understand the context in which one select an OSS component.

First we presented the traditional development processes usually used to develop proprietary software. The development processes used to create OSS components is described in Appendix C. When integrating COTS or OSS components in a system or application, the development process is more complex because of the need to search, evaluate, select and integrate one or several components.
We can observe that the similarities between COTS-based development process and the OSS-based development process are:

- Select a OTS component
- Learn and understand the OTS component

A common role in both development processes is a knowledge keeper of the OTS component.
The observed difference between the two development processes is: Vendor interaction (of technical, administrative and commercial kinds) for COTS-based processes versus no vendor interaction for OSS based process.

4. OTS Selection Processes

In a COTS-based process, as well as in an OSS-based process, an effective and efficient COTS product selection process is essential to the delivery of the full potential of CBS development. [H.K.N.Leung2003].
Most studies focus on proposing selection processes and methods to select COTS components. Since both COTS and OSS components are acquired from third-parties, the users usually have no control of the provided functionality and evolution of these components [Li2006]. They believe that most of the proposed process improvements in COTS component-based development can also be used in OSS component-based development.

Everywhere in the report we will refer to the term "selection process" because we observed from the pre-study that often few components are searched, which then are thoroughly evaluated, and at the end one or several are selected and integrated into the final system. When we say selection process, we refer in fact to the whole process of search, evaluation, selection and integration.

4.1 Phases of the OTS Selection Process

We start this section by presenting the phases of the COTS selection process and the phases of the OSS selection process.

Even if this thesis is about open source components, we present the phases of the selection process of both COTS and OSS because we believe that they share properties.

4.1.1 Phases of COTS Selection Process

According to [Kunda-Brooks1999], there are three phases of a COTS software selection:

Evaluation criteria definition: The criteria definition process essentially decomposes the requirements for COTS into a hierarchical criterion set. The criteria include component functionality (what services are provided), other aspects of a component's interface (such as the use of standards), and quality aspects that are more difficult to isolate, such as components reliability, predictability, and usability.
In [Boehm00], four guiding principles tailor a requirements strategy. These ensures that the requirements are *value-*, *shared-vision-*, *change-*, and *risk-driven*. The *value-*requirements will show how the system will add value for each stakeholder. The *shared-vision* requirements help all the stakeholders quickly to adapt to the new situation, while the precise requirement specifications take more effort to change. The *change-driven* requirements are important because the requirements change in time; it is also important to capture the rationale besides decisions, not only the final decisions. Still it is need for a detailed requirements specification to see possible risks (*risk-driven* requirements). The author also points out that developers need some guideline principles rather than inflexible rules to help each project to determine the best combination of requirements and flexibility for each technical and organizational situation.
COTS projects must also accept *flexibility in requirements* [Morisio et al.2000]. At the moment a COTS is selected, some requirements are immediately satisfied, same other requirements become easy to implement, and others become difficult if not impossible to obtain. In other words, the selected COTS component drives the requirements to at least some extent.

Identification of candidate components: The identification of candidate components, also known as alternatives identification, involves the search and screening for COTS candidates that should be included for assessment in the evaluation phase.
If possible, it is timesaving to reuse internally developed components or components used earlier within the company. COTS identification consists of Web search, literature surveys and product reviews, identification of other reusable system components, and recommendations from external sources [Morisio et al.2000].
In general, evaluating and analyzing all the relevant characteristics of COTS candidates take a great amount of time, typically more than the organization has. Therefore, it is both necessary and cost-effective to select the most promising candidates for detailed evaluation [Kunda-Brooks1999].

Evaluation of the criteria for these candidates: There are currently three strategies to COTS evaluation: progressive filtering, keystone identification, and puzzle assembly. In the keystone strategy, products are evaluated against a key characteristic such as a vendor or type of technology. Progressive filtering is a strategy whereby a COTS product is selected from a larger set of potential candidates, in which products that do not satisfy the evaluation criteria are progressively eliminated from the products list. In the puzzle assembly model, a valid COTS solution will require fitting the various components of the system together [Kunda-Brooks1999].

4.1.2. Phases of OSS Selection Process

In [Wheeler2006], it is described a general process for evaluation of open source software/Free software (OSS/FS) programs. This process is based on four steps: identify candidates, read existing reviews, compare the leading programs' basic attributes to your needs, and analyze the top candidates in more depth.

Identify candidates
The process proposes to first look at lists of OSS programs, lists of "generally recognized as mature" or "generally recognized as safe" OSS programs.
Further, Wheeler recommends searching on specialized sites as:

- http://freshmeat.net/
- http://directory.fsf.org/
- http://osswin.sourceforge.net/
- http://www.theopencd.org/
- http://www.koders.com/
- http://sourceforge.net/index.php
- http://savannah.gnu.org/

Another way is to search using search engines, especially Google's specialized searches for Linux and BSD.

Read existing reviews

It is much more efficient first to learn about a program's strengths and weaknesses from a few reviews than to try to discern that information just from project websites. The simplest way is to use a search engine and search for an article containing the names of all the identified candidates. It is also indicated to search for web sites that cover that market or functional area and see if there are published reviews. Three OSS/FS SCM systems got the most discussion in April 2004: CVS, Subversion, and GNU Arch. Market share is thus an important indirect "review" of a product.

Compare the leading programs' basic attributes to your needs
First, it is necessary to read the project web site to get information about the project. Next, it is important to evaluate the project on a number of important attributes: functionality, cost, market share, support, maintenance, reliability, performance, scalability, usability, security, flexibility/customizability, interoperability, and legal/license issues.

Analyze the top candidates in more depth
This step is mostly done the same way for both proprietary and OSS/FS programs. The same attributes as in the previous step are investigated, but people need to spend more effort by actually trying things out instead of quick reading.

[Madanmohan and De2004] developed an open source component selection model, which involves three basic steps: *collection, incubation,* and *revision*.
In the *collection* step, developers search for open source components and explore the environment for new ideas and components. This should be according to their firm's product roadmap.
In the *incubation* step, developers create and evaluate information about the software component. They question the collected information and evaluate component usability through brainstorming and what-if analysis.
In the *revision* step, developers critically revise existing components through social selection. This revision involves revisiting the existing component library, estimating the value of those components, and deciding whether to add or to extend them in order to meet the customer requirements.

4.1.3 Differences between COTS and OSS

The way the steps are performed in an evaluation is different between OSS/FS programs and proprietary programs [Wheeler2006]. Some key differences may be:

- The information available for OSS is usually different from proprietary programs
- OSS can be changed and redistributed by customers. This difference affects many factors such as flexibility/customizability and support options.
- No hidden defects or features in OSS!
- All the features are available in OSS!

When considering open-source based platforms, there are at least two crucial differences when compared to more traditionally proprietary platforms [Derick and West2004]:

- The open source solution uses collaborative R&D and support in cooperation with firms whose role is far less central or defined. The R&D, sales, and support for the proprietary solution are the responsibilities of a well-defined profit-making enterprise.
- OSS source code is widely disseminated to all and thus organizations have the opportunity to modify the software to suit their own needs. Often, the migration of internal changes to the provider is difficult and the changed code ends up as internal responsibility.

4.2 General Considerations and Criteria in an OTS Selection Process

In [Alves and Castro2001], four main dimensions should be considered in a COTS selection process: domain coverage, time restriction, cost rating, and vendor guaranties. Some can apply to OSS as well.

- **Domain Coverage**: The components have to provide all or part of the required capabilities which are necessary to meet essential customer's requirements. The domain coverage can also apply to OSS, since both COTS and OSS must anyway fulfill the customer's requirements.

- **Time restriction:** Software companies usually operate in a very rigid development schedule, on which their competitiveness depends. Selection is a time consuming activity where a considerable amount of effort is necessary to search and screen all the potential COTS candidates. This can also apply to OSS when companies select components. Whether it is COTS or OSS, software companies still have time restrictions.

- **Cost rating:** The available budget is a very important variable. The expenses when selecting COTS products will be influenced by factors such as: license acquisition, cost of support, adaptation expenses, and maintenance prices.

- **Vendor guaranties**: An important aspect to be considered in the selection activity is to verify the technical support provided by the vendor. Some issues have to be taken into account, e.g., vendor reputation and maturity, number and kind of applications that already use the COTS, and the characteristics of the maintenance licenses.

In [Ruffin and Ebert2004], some criteria that OSS must meet are described:

- Build on and follow mature and commonly used industry standards such as de facto standards like Linux.
- Have a strong OSS community.
- Be broadly supported by several Independent Software Vendors (ISVs) for distribution, evolution, and support.
- Have a clear, indisputable legal status regarding Intellectual Property Rights (IPR) and the right to use it.

In [Woods and Guliani2005], the elements of open source maturity (elements that are direct indicators of the potential difficulties that can be encountered when using open source) are discussed in Table 4.

Element of maturity	Description
Leadership and culture	It is very important for an open source project to have good leaders and highly knowledgeable individuals.
Vitality of community	The leadership and the vitality of the community are correlated. In a good community, everyone finds something useful to do, and there is a division between the project's developers and users. The size of the community is also a good indicator of a project's viability.
Quality of end-user support	Active forums, well-maintained FAQ, and documentation that is available through search engine can save a lot of time.
Extent and scope of documentation	The quality of documentation is another good clue about a project's work process. It is normal to expect that the instructions for installing, running, and fine-tuning a piece of software to be written in clear English.
Quality of packaging	It is a good indication of maturity of a project to have an installation package that can install the software easily on many different platforms and configurations.
Momentum	Often releases indicate that the project is active; the users do not have to wait too long for bug fixes. It is important to check the release history to see if the new releases are mainly significant or more trivial.
Quality of code and design	The quality of code and design are important technical issues to consider anytime.
Quality of architecture	The quality of architecture is also an important technical issue.
Testing practices	Some open source code come with automated, built-in testing facilities as standard features. The presence of unit tests is a key indicator of good design.
Integration with other products	All software in the enterprise operates as part of an *ecology*, meaning that a set of interdependencies cause programs to call on each other.
Support for standards	The programs need to use standards-based APIs. The problem of dependencies and nonstandard APIs is sometimes addressed in open source projects by having a program download with a pre-selected set of other programs (the programs that make up its local ecology).
Quality of project site	A good project site can attract more people.
License type	The main license types will be explained in the next sub-chapter
Corporate commitment	Several open source projects, such as the Linux operating system and the Apache Web Server, have enjoyed support

	from large, established computer companies, including IBM, Sun, HP, and Dell.

Table 4: Elements of open source maturity

Some maturity models exist to help organizations to successfully implement open source software. An example is the Open Source Maturity Model (OSMM) from Navica [Navica]. OSMM assesses the maturity level of all key product elements: a) software, b) support, c) documentation, d) training, e) product integration, and f) professional services.

[Li2006] shows that:

- Formal selection processes were rarely used
- Functional completeness was regarded to be more important than architectural compliance in COTS component selection
- The actual OTS-based development process was the traditional process with OTS-specific activities. The development process was dominated by existing company/department rule instead of the decision of using OTS components.
- The actual OTS component selection can be done in different phases. The main evaluation processes are familiarity-based or hands-on-trial-based. The phase to select OTS component depends on the project members' familiarity to components.

Since the results above are based on a large international survey, it is interesting to compare them with our results.

[Hauge and Røsdal2006], "A Survey of Industrial Involvement in Open Source" shows that:

- Open source components are primarily regarded as commodity software
- High availability of components, source code, and information related to the components are the most important motivations for using OSS components
- Selection of OSS components is primarily done as a process based on previous knowledge, informal searches, and subjective evaluations of the components.

The following company roles were investigated in the thesis mentioned above: open source owner, open source participant, inner source participant, and user of open source components. User of open source components means a company which develops software using open source components and tools. Our role (i.e., integrator of open source components) is a bit different but it is anyway interesting to compare the results since approximately the same number of respondents was used in both studies.

An exploratory study based on structured interviews performed in India and US between June and September 2003 [Madanmohan and De2004], shows that:

- The teams had no need to look at or modify the source and did not have enough resources (knowledge, skills, or manpower) to do so.
- If the open source component offers the best solution and reliability for the price, then it is the most appropriate.
- From the perspective of a commercial firm, the license must explicitly permit the distribution of software build from modified source code.

The results of [Li2006] and [Hauge and Røsdal2006] are related to our work since they have also used the "integrator of open source components" as company role.
The results of [Madanmohan and De2004] are also related to the motivations to integrate OSS components, which is of interest for us as well.

From these different sources, we observe that there are three different segments to be considered when selecting open source components:

1. Business aspects
2. Technical aspects
3. Organizational aspects
4. Professional aspects

4.3 COTS Selection Methods

According to [H.K.N.Leung2003], there are three categories of COTS product selection methods: the *intuition* approach, the *direct assessment* approach and the *indirect assessment* approach. In the *intuition* approach, software developers select COTS products according to their experience and intuition, it's a subjective approach. Most of the methods belong to the *direct assessment* (DA) approach, which selects COTS components directly from their source. The efficiency of the direct assessment approach is inversely proportional to the product of the number of modules in the system to be developed and the total number of modules in the candidate COTS products. The problem with these approaches is that they can be quite inefficient when there are many components to be investigated. The method proposed in [H.K.N.Leung2003] (i.e., the Domain-based COTS-product selection method) is different from the other methods based on the intuition and the direct assessment approach because it uses domain models to avoid the matching among all the available COTS components in the marketplace, and only compares the COTS that belong to the domain that is represented by the model, so they called the method as "indirect assessment".
A range of *direct assessment* methods have been proposed for COTS product selection: [Alves-Castro2001], [Kontio1996], [Maiden and Ncube98], [Morisio et al.2000], [Chung and Kooper2004], [Tran et al.1997], [Fox et al.1997].

OTSO
Kontio [Kontio96] proposed the Off-The-Shelf-Option (OTSO) selection method. This method is based on direct assessment.
OTSO assumes that the requirements of the proposed system already exist. But in practice it is possible that the requirements cannot be defined precisely because the use of some COTS products may require some changes to the requirements. The main principle

followed by OTSO is that of providing explicit definitions of the tasks in the selection process, including entry and exit criteria. The inputs are: requirement specification, design specification, project plans, and organizational characteristics. The outputs are the selected COTS product, the results of the evaluation, and cost models.
OTSO comprises three phases: searching, screening and evaluation. The searching phase attempts to identify all potential COTS candidates. In the screening phase, it is decided which COTS candidates should be investigated further. In the evaluation phase, COTS candidates undergo a detailed evaluation. Criteria for COTS selection are:

a. functional requirements of the COTS
b. required quality-related characteristics
c. business concerns
d. issues relevant to the software architecture

The OTSO method uses the Analytic Hierarchy Process (AHP) to consolidate the evaluation data for decision-making-purposes. AHP is based on the idea of decomposing a complex, multi-criteria decision-making problem into a hierarchy of the selection criteria. It helps decision makers to structure the important components of a problem into a hierarchical structure. The method assumes that the requirements exist; they are used to define the evaluation criteria.

CAP
[Ochs et al.2001] created a COTS Acquisition Process (CAP) method. CAP is based on OTSO, but is strictly measure-oriented, allowing for the evaluation towards cost-efficiency and systematic changes. It consists of three components:

- The CAP Initialization Component (CAP-IC), which consists of: the identification of criteria, estimation of effort needed to apply all evaluation criteria, set up pf the measurement plan, and a review step that certifies that all CAP-IC activities have been conducted correctly.
- The CAP Execution Component (CAP-EC), which executes the measurement plan developed in Cap-IC by trying to achieve highest possible efficiency.
- The CAP Reuse Component (CAP-RC). The only activity here is to package and store all useful information generated during the enactment of a CAP project.

CISD
Tran, Liu and Hummel have proposed the *COTS-based Integrated System Development* (CISD) model [Tran et al.1997]. This method is based on direct assessment.

The inputs for the selection process are the system requirements and information about the COTS products, and the outputs include a prioritized list of COTS products and the architecture of the system.
CISD consists of three phases: identification (with two sub-phases: product classification and product prioritization), evaluation and integration.
In the evaluation phase, three attributes of the COTS products are examined: functionality, architecture, and performance. The integration phase encompasses all of the

development effort that is required to interconnect the different selected COTS products into a single integrated system.

PORE

The *Procurement-Oriented Requirements Engineering* (PORE) method [Maiden and Ncube1998], [Maiden et al.2002] integrates techniques from requirements engineering, knowledge engineering, multi-criteria decision-making and feature analysis to guide the selection of COTS packages. It proposes a concurrent development process, in which stakeholders' requirements acquisition and COTS package selection occur at the same time. This method is based on direct assessment.

The method uses a progressive filtering strategy, whereby COTS products initially selected are then progressively eliminated when they do not satisfy the evaluation criteria. The inputs of the method are the attributes of the COTS products, supplier requirements, and information about product branding, open standards, product certification, the development process, reliability, security and dependability. The output is a shortlist of COTS products. In this method it is not clear how requirements are used in the evaluation process and how products are eliminated.

CRE

The COTS-based Requirements Engineering (CRE) method [Alves-Castro2001] was developed to facilitate a systematic, repeatable and requirements-driven COTS product selection process. The method focuses on non-functional requirements to assist the process of selection of COTS products. This method is based on direct assessment.

The method has four iterative phases: identification, description, evaluation and acceptance. The identification phase is based on a careful analysis of influencing factors; there are five groups of factors: user requirements, application architecture, project objectives & restrictions, product availability and organizational infrastructure.
During the description phase, the evaluation criteria are elaborated in detail.
In the evaluation phase, a particular COTS product is selected based on estimated cost versus benefits.

The selection criteria for the CRE method are:

- Domain coverage:
  Both functional and non-functional requirements are important to meet customer needs. A special focus is on non-functional requirements.
- Time restriction:
  The time available for searching and screening all the potential COTS candidates is limited.
- Cost rating:
  Acquiring a license, support cost, expenses associated with adapting the product, maintenance cost should all be within the available budget.
- Vendor guaranties:
  The technical support provided by vendor is also important.

A disadvantage of the CRE method is that the decision-making process can be very complex.

IIDA

Another work presented in [Fox et al.1997] describes the *Infrastructure Incremental Development Approach* (IIDA) for the development of technical infrastructure using COTS products. This approach is a combination of the classical waterfall and spiral development models in order to accommodate the needs of CBS development. The process of selecting COTS products in the IIDA relies on two phases: *analysis prototype* and *design prototype*. In the analysis-prototype phase, COTS candidates are selected from each COTS product family. In the design prototype phase, the best COTS products from the earlier phase are selected and evaluated. Basic evaluation criteria include functionality and performance.

CARE

The *COTS-Aware Requirements Engineering* (CARE) process is described in [Chung and Cooper2004]. This method is based on direct assessment.

The CARE approach is characterized as agent-oriented, goal-oriented, knowledge based, and has a defined methodology, or process. The goal is to define a methodology that supports the definition and selection of COTS components from a technical view. The early artifacts for the system under development are accounted for including:

- The agents
- Soft-goals: non-functional goals that are achieved not absolutely but in a "good-enough" sense)
- Hard-goals: functional goals
- System requirements
- Software requirements
- Architectural elements

The traceability relationships among the artifacts are also established and maintained (e.g., a soft-goal is refined and traced to specific system requirements).

All these descriptions are information like that found on marketing brochures for existing products. These general descriptions are used to determine if the product appears to be potentially useful.

This methodology departs from the description of the COTS components stored and maintained in a knowledge base, or repository. As the goals are defined, analyzed, and negotiated, the requirements engineer (RE) searches the repository for COTS components that appear to be (possible) matches. As the system develops, goals are defined into requirements and the RE determines if the identified components still seem suitable or if other components seem to be a better fit.

In spite of this methodology pretends to improve reuse by means of a repository of COTS descriptions, it presents three obvious disadvantages: (1) it is not clear how to build this repository; (2) the maintainability of the repository is especially difficult taking into account the rapid evolution of the information of each product and the increasing amount

of products in the market and this aspect is not considered, (3) the searching process is not very efficient having to look for components in a widespread range of descriptions.

DBCS

[Leung03] have developed an indirect method: the Domain-Based COTS-product Selection (DBCS) method, which makes use of the specific domain model to decide the suitability of the COTS product. This approach has the goal to reduce the amount of work required for the selection process. There are two basic strategies for the selection of a COTS product; depending on whether an application development needs the best available COTS product: best-fit strategy and first-fit strategy.

GoThIC

The GoThIC (Goal-Oriented Taxonomy and reuse Infrastructure Construction) method [Ayala and Franch2006] was designed to drive the construction of OTS domain-knowledge repositories populated with OTS components.

The GoThIC method has been structured into seven activities with the ultimate goal of the method to populate a knowledge base with data. The resulting artifacts can be seen as a friendly and flexible taxonomy and knowledge base that can support the OTS component selection process. The method deals only with component searching and is not intended to address the whole OTS component selection process. Therefore, the method is shortly presented here but not in Table 5.

Table 5 : Summary of methods dealing with COTS selection

Methodologies of Process	Input	Selection procedure	Selection criteria	Reuse of knowledge	Descriptive criteria definition
OTSO	- Requirements Specifications - Design Specifications - Project plans	-Searching -Screening -Evaluation	-Functional requirements -Quality characteristics -Business concerns -Relevant software arhitecture	-	√
CAP	- input from Requirements Engineering (RE)/System Design(SD) in form of requirement specification documents for	- The CAP Execution Component (CAP-EC): executes the measurement plan developed in Cap-IC. - The CAP Reuse Component (CAP-RC): the only activity is to package and	The CAP Initialization Component (CAP-IC), which consists of: the identification of criteria, estimation of	√	√

	the respective system component.	store all useful information.	effort needed to apply all evaluation criteria, set up of the measurement plan.		
CISD	- System requirements - COTS Products	-Product identification * Classification * Priorization -Evaluation	-Functional - Architecture/inter-operability -Fulfill multiple requirements from the service domain	-	√
PORE	- COTS Product attributes - Supplier Requirements - Product development process	-Acquires customer requirements -Multi-criteria decision making -Rejects non-compliant COTS -Explores other candidates for new customer reqs.	-Development process -Supplier CMM level - Product/supplier past record -Reliability -Security -Dependability	-	*
IIDA	- COTS from each COTS product family	-Analysis prototype -Design prototype	-Functional requirements -Performance	√	*
CARE	- Soft-goals: non-functional goals that are achieved not absolutely but in a “good-enough” sense) - Hard-goals: functional goals - System requirements - Software requirements - Architectural elements	-Select candidate hard-goals -Preliminary component matching -Detailed component matching -Select component	-The agents -Soft-goals -Hard-goals: -System requirements -Software requirements -Architectural elements	√	√
DBCS	- The specific	- Best-fit strategy or	-Makes use of	*	√

	domain model of the intended system to decide the suitability of the COTS product.	- First-fit strategy	the specific domain model of the intended system.		

(√) addresses the issue fully (*) deals with the issue but not fully(-) means not deal with the issue

The efficiency of the direct assessment approach is inversely proportional to the number of modules in the system to be developed and the total number of modules in the candidate COTS products. The cost of selecting an appropriate COTS product can be expensive and hence, may offset the advantages of using COTS [Leung03]. We think that the same applies for OSS components; that it is very important to select OSS components in an effective way. We have presented the different methods dealing with COTS selection in order to understand them, to be able to propose questions for the interview guide, and then to compare our results with these methods described in literature.

4.4 Summary

The first three parts of the chapter presented the phases of COTS selection process and the phases of an OSS selection process.

Even if this thesis is about open source components, we presented the phases of selection process of both COTS and OSS. We believe that the phases of the COTS selection process can also apply to OSS in general, even if there are some differences, e.g., in the OSS selection process, components are not evaluated against a key characteristic such as a vendor.

[Wheeler2006] and [Modanmohan and De2004] describe specific phases for the OSS selection process. Some phases are also common for COTS. For example, the incubation phase for OSS described in [Modanmohan and De2004] is similar with the "identification of candidate components" phase for COTS described in [Kunda-Brooks1999].

From all the sources described in 4.1, we can propose an OTS selection process:

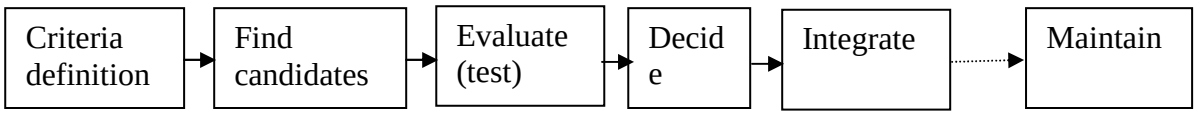

Figure 3: Phases of OTS selection process

The general considerations and criteria in an OTS selection process are presented both for COTS and OSS. Again, it is interesting to understand the differences and the similarities between these two.

Finally, the COTS selection methods and a summary of methodologies dealing with COTS selection close this chapter.

Part 3 - Research

5. Research Agenda

This chapter presents the main empirical strategies with focus on a partly explorative and partly descriptive survey, since this is the main research method used in our project. We then present the research questions, interview definition, and the research design, followed by the research context.

The main activities of this master project are:

- Project start-up;
- Further elaboration of existing literature;
- Improve the interview guide based on the comments and feedback from the depth study project; we have re-write some questions, some were taken out and some others were added.
- Prepare a list of Norwegian companies (IT companies and non-IT companies which have an IT department), group the companies in different categories to be able to use stratified random sampling;
- Randomly choose a few companies from each strata and take contact for an interview;
- Company visit and interviews;
- Analyze and present data and results;
- Project deadline.

5.1 Empirical Strategies

There are two types of research paradigms that use different approaches in empirical studies:

- Qualitative research – is concerned with studying objects in their natural setting. It is concerned with discovering causes noticed by the subjects in the study, and understanding their view of the problem at hand.
- Quantitative research – is mainly concerned with quantifying a relationship or to compare two or more groups. The aim is to identify a cause-effect relationship.

In the empirical software engineering field, there are three major types of investigations (strategies) [Robson93]:

- ***Survey:*** A survey is often an investigation performed in retrospective, when, for example, a tool or technique has been in use for a while. The primary means of gathering qualitative or quantitative data are interviews or questionnaires. These are done through taking a sample which is representative from the population to be studied. The results from the survey are then analyzed to derive descriptive or explanatory conclusions. They are then generalized to the population from which the sample was taken.

- ***Case study:*** A case study is conducted to investigate a single entity or phenomenon with in a specific time space. The researcher collects detailed information on, for example, one single project during a sustained period of time. Case studies are used for monitoring projects, activities, or assignments.
- ***Experimentation:*** Experiments are normally done in a laboratory environment, which provides a high level of control. The objective is to manipulate one or more variables and control all other variables at fixed levels.

Surveys and case studies can be both qualitative and quantitative, while the experiment is most often quantitative.

The general objectives for conducting a survey is any of the following [Babbie90]:

- ***Descriptive:*** Descriptive surveys can be conducted to enable assertions about some population. This could be determining the distribution of certain characteristics or attributes.
- ***Explanatory:*** Explanatory surveys aim at making explanatory claims about the population. For example, when studying how companies select open source components, we want to explain why some prefer a special technique while others prefer another. By examining the relationship between different techniques and several explanatory variables, we may try to explain why developers choose one of the techniques.
- ***Explorative:*** Explorative surveys are used as pre-studies through investigations to assure that important issues are not overseen. Creating a loosely structured questionnaire and letting a sample from the population answer it could do this. The information is gathered and analyzed, and the results are used to improve the full investigation. In other words, the explorative survey does not answer the basic research question, but it may provide new possibilities that could be analyzed and should therefore be followed up in a more focused or thorough survey.

5.1.1 Choice of Research Methods

The study in this thesis will be performed as a partly explorative and partly descriptive survey, because there were no appropriate research questions based on the literature study, and because we want to discover new knowledge about selection of open source components, as outlined in the research questions. The purpose of this study is therefore to answer the research questions by performing an explorative/descriptive study to find new knowledge about selection of open source components and to find support for the findings in the depth study project.

When designing a research study, [Cooper and Schindler2001] recommend using a two stage research design for explorative studies. The first stage is the most explorative; where the purpose is to define research questions and develop a research design. The second stage is more descriptive and is performed on a larger sample of the population.
In the autumn of 2006, we performed the first stage, where a structured interview was performed on 7 subjects. The interview was chosen as the method because it was the best trade-off between usage of time, and its ability to provide detailed textual description for the research questions. The interview was structured as much as possible in order to get

comparable results from the respondents. The interview had open-ended questions because of the explorative research question.

5.1.2 Population and Sampling Methods

In this thesis, we have performed the second stage of the explorative study and try in addition to find support for the observations and findings in the depth study. We have interviewed nine respondents (IT companies). The respondents were drawn randomly from a rather large list of Norwegian IT companies. The list of Norwegian companies (IT and non-IT which are doing software development) was assembled in order to make it possible to use stratified random sampling. We have used the following methods to create the list of Norwegian companies:

- Re-used a list with IT companies compiled by [Li2006] as part of his sample. This is about 15% of the total list we assembled. A more detailed description of this starting list is presented in below ;
- Search in "Yellow pages" to find Norwegian IT companies;
- Look-up the LinkedIn connections/contacts of the supervisor Dr. Carl-Fredrik Sørensen to find Norwegian companies. The primary contacts either presently work at a company, or may have previously worked in other companies; the ability to also look at the connections of the primary contacts, resulted in a snowball effect to enlarge the population since new companies could be found by following the links one level further than the original contact list. The original contacts of the supervisor were approximately 200 with the possibility to reach about 9000 more through the extended network.
- Used magazines (e.g. ComputerWorld and Teknisk Ukeblad) and newspapers to find Norwegian companies that advertise to employ software developers (this is an indication that these either are IT companies or that they at least have a software development unit).

Below is a short description of the list used in previous studies. The list was originally provided by ICT-Norway and later compiled and quality-controlled by Dr. Jingyue Li [Li2006]:
The list was in the form of an Excel file with several sheets:

- A sheet with the 50 largest general companies. This is a list with the largest Norwegian companies (not necessarily IT companies) grouped by focus. Mainly the company names were provided. We have used this sheet by adding all these companies' names to our own spreadsheet containing all the companies.
- A sheet with the 100 largest IT companies. We have used this sheet by adding all these companies to our own spreadsheet.
- A sheet of companies with size between 20 and 99 employees. We excluded the dead or merged companies or without software development, and included the rest of companies.

- A sheet of companies with size between 1 and 19 employees. We excluded the dead or merged companies or without software development, and included the rest of companies.

We replicated the stratification approach of companies from this list. The list was not up-to-date, thus we had to check the status of each company to see if the company still existed, to find out the actual number of employees, and whether a company had migrated from one stratum to another.

Totally, the list ended up with about 620 Norwegian companies. Many of these are not purely Norwegian companies since some are foreign companies with an IT department or sales filial in Norway. The Web was used to find out the main focus and the size of the companies, and whether they are an IT company or not. After this first step we were able to group the Norwegian companies as indicated in the Table 6:

IT or non-IT company	Size
250 non-IT companies which have a software development unit.	All sizes.
75 IT companies	More than 100 employees.
103 IT companies	Size between 20 and 99 employees.
116 IT companies	Size between 1 and 19 employees.

Table 6: Norwegian companies grouped by size

The total number of companies in the Table 6 is 544. The difference between 620 and 544 are companies without published or searchable information (e.g. address, web page) or companies that merged or just disappeared.

For many of the companies described in Table 6, we were unsure about the size and whether they are integrating OSS components into their systems or applications.

Therefore the next step (done by my supervisor) was to send emails to most of the companies and ask for information about the size and whether they are integrating/using OSS components or not. About 70% of the companies answered the email and provided information about the company size and whether they are integrating OSS components into their systems/applications or not. After this second step, we were able to group the Norwegian IT companies as indicated in the Table 7:

Number of the IT companies in the group	Size	Response rate to the email	Do integrate OSS
75	Larger than 100 employees	56%	22 of 36
103	Size between 20 and 99 employees.	83%	20 of 99
116	Size between 1 and 19 employees.	84%	47 of 97

Table 7: The distribution of the Norwegian IT companies integrating OSS components.

We took contact with more than 3 companies in each list until we ended up with three respondents in each group (4 contacts in the list with big companies, 5 contacts in the list with medium size companies and 6 companies in the list with small companies).
We randomly selected three from each of the three groups, and ended up with three respondents in each group/stratum (totally nine respondents) selected for interviews.

5.2 Research Questions

The goal of this project is to investigate state-of-the-practice of selection of OSS components from both the organizational and the process point of view. The motivation is to give guidelines on how to select and evaluate an OSS component, based on experiences and lessons learned from finished projects.
A pre-study was performed with the goal to find out what are the actual processes and selection processes used in OSS component-based development projects in 2006, followed by a somewhat larger and more general study in this thesis.

Research goal:
Explore the following company role: *integrator of open source components*.
We want to discover what the selection process is when companies want to integrate OSS components into a system or application. The component are referred to should neither be infrastructure OSS (e.g., Linux, MySQL, or Apache Web server) nor open source tools (e.g., Eclipse, PHP, Perl).
Further, we want to explore their development processes with focus on selection processes of OSS components.

Company role description: Integrator of Open Source Component is typically a company which develops software using open source components.
Some high-level activities directly related with the OTS selection processes and the most relevant requited roles are described in Table 8 [Ayala et al.2006].

Activity	OTS Users Role
Finding Candidate OTS	Market Watcher (MW)
Evaluating OTS Candidates	Quality Engineer (GW)
Selecting OTS Component	Selector (S)
Documenting the decision	Knowledge Keeper (KK)

Table 8: Activities and roles in OTS components selection

We explore all these OTS company roles in our study. In fact, our research questions try to relate these roles to the company size.

Research questions:
RQ1: Who (which role in a company) initiates and performs the work related to OSS harvesting and when in the development process?

RQ1.1: Who (which role) initiates the OSS harvesting in a company?

RQ1.2: Who (which role) does the work related to the harvesting?

RQ1.3: Who (which role) takes the final decision about integrating an OSS component?

RQ1.4: When in the development process does one select OSS-components?

RQ2: What are the motivations for and experience of using OSS components?

RQ3: What is the current process of selecting OSS components?

RQ4: How to find OSS components?

RQ5: What are the evaluation criteria when selecting OSS components?

RQ6: What versions are considered and how to deal with new versions?

RQ7: How to maintain the knowledge about the selection processes and the knowledge about the selected OSS components?

5.3 Interview Definition

Conducting an experiment is a labor-intensive task. In order to utilize the effort spent, it is important to ensure that the intention with the experiment can be fulfilled through the experiment. In the definition phase, the foundation of the experiment is determined. The purpose of this phase is to define the goals of an experiment [Wohlin et al.2000].

5.3.1 Object of Study

The object of study is the process of selection of open source components in industry. The interviews want to explore what are the practices in the industry. This empirical study is only explorative, but the final goal is to find out practices that can be used as input in the decision-making in any improvement seeking software organization [Wohlin et al.2000].

5.3.2 Purpose

The purpose of the interviews is to evaluate the process of selection of open source components within different companies to find possible deviation in process between smaller and larger companies.

5.3.3 Quality Focus

The quality focus is the effectiveness of the process. It is very important to discover an effective way to select components, and that would be very useful for the software industry.

5.3.4 Perspective

In this empirical study, the perspective is from master student and researcher's point of view. The interviews are performed by one master student (see Appendix D and E for the interview guide and a help for the interview guide) but under the supervision of a researcher. Also, other researchers contributed to the interview guide with useful comments.

5.3.5 Research Context

The depth study was performed in the context of the Norwegian COSI project. A more detailed description of this research context was presented in Chapter 1. However, this master thesis has been performed as separate research.

5.4 Interview Planning

The interview definition presented in the previous section is input for the interview planning. The output of the interview planning is the interview design.

The interview is designed to provide qualitative answers to the research questions. The design of the interview is explorative with the main objective to discover new aspects of open source, aspects described by the research questions. In addition, we want to find support for the findings in the depth study where applicable.

5.4.1 Context Selection

The interview is performed by a student, but under the supervision of a researcher. The interview addresses real problems in the software industry. However, because of the limited number of respondents, the results are not valid to a general software engineering domain.

5.4.2 Respondents

The interviews were conducted at nine Norwegian software companies, where IT managers, software architects, or developers were interviewed.
The companies were selected using *random sampling within strata.*
The selected companies have the specified role in open source, i.e., integrator of open source components.

5.4.3 Interview Design

The interview guide used is presented in the Appendix D. We have created a help description for the interview guide. This description was not sent to the subjects but mainly used as a supporting instrument during the interviews. Both the interview guide and the help for the interview guide were written in English, even if it were only used to interview Norwegian software companies.
The interview guide contains both open questions and closed questions with predefined answers to choose from.
During the interview it was possible to ask follow-up questions to further elaborate on areas where more information was needed.

5.4.4 Interview Limitations

The number of interviews was limited to nine Norwegian software companies and this is a clear limitation for external validity and thus generalisation. Performing interviews is a time consuming task and it is thus difficult to interview a representative sample of responding companies. Therefore, the validity of the results will be discussed based on the limited number of responding companies.

Many problems might occur when designing and performing an interview. Some problems are related to the sample of respondents answering the interview; other problems are categorized as non-response or response errors. Non-response errors occur when the respondent for some reason will not answer the interview. This reduces the sample size and has impact on the validity of the results. Response errors occur when the reported data does not match the actual true data.

5.5 Collecting Evidence

We will here discuss how we collected the data and information from our interviews.
Totally, it was conducted nine interviews. The companies interviewed were Visma, Sirius IT, TietoEnator, WebOn, Abeo, DKDigital, Commitment, Grieg Multimedia, and Riventi.
The interview guide was in English and therefore we conducted the interviews in English. To allow the interview objects to be prepared for the interview, we sent them the questionnaire a few days before the interview. All the interviews were done by phone. Each interview was recorded , in addition the interviewee marked the answers directly on the paper version of the interview guide.
Some interview objects started to talk about a specific question without being asked directly, some were asked.
The average time spent on one interview was about 40 minutes.

5.6 Analyzing the Evidence

When the interviews were finished, we analyzed the data by reading the answers but also listening to the audio records. We found both qualitative and quantitative results.
To back-up our findings, we included some statements in the thesis as quotes.
We have analyzed the results also taking into account the findings from the previous survey performed in the depth project.

5.7 Summary

In this chapter, we presented a summary of the empirical strategies with focus on an explorative and descriptive survey.
Defining the research goal and research questions is an essential part of the chapter. We have defined a company role: *integrator of open source components*. The integrator of Open Source Components is a company which develops software using open source components.
We have defined our research questions by identifying research problems in the literature, but we have also considered what would be beneficial for the software industry. The interview definition and the interview planning are necessary steps before performing the interview.

Part 4 – Results, Discussion and Conclusions

6. Results

This chapter starts by presenting the link between each research question and the corresponding questions from the interview guide in Table 9. For example, research question 1 is answered with the help of the first four questions from the interview guide.

Research question	The corresponding questions in the interview guide
1	1, 2, 3, 4
2	5
3	6,7
4	8
5	9-13
6	14-16
7	17-21

Table 9: The corresponding questions in the interview guide for each research question

6.1 Create Descriptive Findings

This study is based on an explorative research goal. Since we only interviewed nine respondents, the results cannot be used to test hypotheses, but rather be used to build up theories and to create hypotheses that later can be tested on a more representative sample. Therefore, several descriptive findings will be presented based on the results from the research questions. These descriptive findings will be discussed later versus the findings obtained in the depth study from autumn 2006, and also versus the literature review. Some of the findings should be checked by further studies using a larger sample.

6.2 RQ1: Who (which role in a company) initiates and performs the work related to OSS harvesting and when in the development process?

RQ1.1: Who (which role) initiates the OSS harvesting in a company?

RQ1.2: Who (which role) does the work related to the harvesting?

RQ1.3: Who (which role) takes the final decision about integrating an OSS component?

RQ1.4: When in the development process does one select OSS-components?

This research question is answered by the first 4 questions of the interview guide. We have asked these first questions as open questions, without giving any alternatives.
Regarding the initiation phase, most respondents answered that is the *software developer* who does it. Also most respondents answered that it is the *software developer* who does the work related to harvesting.
The final decision is mostly taken by either a s*oftware developer* or a *leader*. A leader can have one of the roles: chief executive, project owner, company leader group, or project architect. As indicated by a respondent from a large company: *"the project team makes a*

recommendation and is the Chief Executive Manager who has the final word about integrating OSS components. The customer is also involved." In one medium size company, *"it is the software developer who takes the final decision for integrating small components and the IT manager for big components."*

A significant difference was observed between the company sizes:
In all the small companies we interviewed, it is the software developer who initiates and does the work related to harvesting, and also takes the final decision about integrating OSS components.
In the medium and large companies much more roles are mentioned. The person who initiates and performs the work related to harvesting in addition to the software developer can be: project manager, project team, architect, and system integrator. The person who takes the final decision in addition to the software developer can be: chief executive manager, project owner, the company leader group.

DF1: In small companies, it is the software developer who initiates and does the work related to component harvesting, and takes the final decision about integrating OSS components. More roles (in addition to software developer) such as: project manager, project team, architect, system integrator, company leader, are associated to this process in medium and large companies.

When asking the question number 4 ("When in the development process does one selects OSS components?"), we have observed differences between the large and small components, especially in the case of large and medium size companies. The large and medium size companies tend to select the large components at an early stage (requirements/analysis) while the small components can be selected during the development phase. The small companies do not differentiate too much between large and small components; they usually select OSS components in the development phase. *"For small components, it is more an ongoing process, it can happen really anytime"*, said one of the respondents from a large company. Therefore we create the following descriptive finding:

DF2: Large components are usually selected at an early stage during the development process, while small components can be selected anytime during the development process.

6.3 RQ2: What are the motivations for and experiences of using OSS components?

There are many motivations to integrate OSS components into a system or application. All the three large companies plus some medium and small companies have mentioned quality, shorten time to market, and cost as the principal motivations for integrating OSS components. *"Our motivation is to get something we know is of high quality and we want to get it fast. We do not want our developers start writing programs from scratch. At least in the Java world we find very often that the highest quality libraries are the ones that are open source and not the commercial. The money has very little to do with the decision"*, said one of the respondents from a large company.

Other motivations are: full access to the source code, it is flexible to integrate OSS components; it is a kind of philosophy in the company to use OSS instead of COTS, it is recommended by partners, and it is fun to work with OSS.
An interesting observation from a respondent from a small company is: *"The principal motivations to integrate OSS components are different depending on the view point. From the business point of view, the motivations are shorter time to market and price. From the developer point of view, the motivations are: documentation, performance, and flexibility to integrate."* The other companies did not mention different view points.
We could not observe any particular differences between the three groups of companies.

DF3: Higher quality, shorter time to market, and cost are the principal motivations to integrate OSS components into a system or application.

6.4 RQ3: What is the current process of selecting OSS components?

Most of the companies do not use a formal process for selection of OSS components. Even if there is a process for selection of OSS components, it is not a formal one. Only one respondent from a large company said that: *"we have a general development process which includes the process for selection of OSS components. We use Rational Unified Process (RUP), which contains a lot of methods."*

Regarding the main activities of the OSS selection process, the answers were quite spread because of the opened question. Below are listed the different indicated activities:

- Define criteria;
- Establish functional and non-functional requirements;
- Search components internally in the company;
- Check the Internet for libraries or components;
- Check major OSS web sites like SourceForge and the popularity of the OSS components;
- Go briefly to forum discussions;
- Check the maturity of the component;
- Check if the company has the necessary competence to integrate the OSS component;
- Evaluate the components, and the consequences of using them;
- Check licenses;
- Test;
- Integrate.

Some companies have already good knowledge about the existing OSS components, so they do not need to do a real search every time. One respondent from a medium company said: *"Selecting an OSS component is a matter of using people experience and knowledge about OSS components. Usually we know about OSS components, this is part of our strength."*

We have observed that the large companies perform more activities than the medium size companies and that the medium size companies perform more activities than the small companies.

DF4: Companies do not use any formal processes for the selection of OSS components.

6.5 RQ4: How to find OSS components?

The different kinds of ways to find OSS components in companies are:

- Use a repository containing components used before;
- Search on Google, specialized web sites, or at OSS communities;
- Use mailing lists;
- Read articles, magazines;
- Read forums and news groups;
- Check the most popular OSS products;
- Recommendations from partners and suppliers;
- Check experiences from other projects.

However, the most common ways to find OSS components are to use a repository with components used before or to search Google, specialized web sites or OSS communities.

DF5: Companies usually find OSS components by searching repositories containing components used before, by Web search (Google), specialized web sites, and OSS communities.

6.6 RQ5: What are the evaluation criteria when selecting OSS components?

We have grouped the evaluation criteria as technical issues, organizational issues of the OSS community, and professional issues. We think that all these are important. Because the OSS components always have a community around them, it is interesting to discover what organizational issues of the communities companies are considering. The companies may have or not have skilled people to change the open source code if necessary, the fact that companies have heard about some OSS components may also influence the selection process. Therefore, it is also important to discover what professional issues companies consider when evaluating OSS components.
This research questions is asked by the opened questions 9 to 13 in the interview guide.
When asking question 9 (“What technical issues is your local business unit considering when selecting an OSS component?”), the most relevant answers are: a) standards compliance, b) matching functionality, c) stability, d) good quality (design, architecture, and documentation in the case of big components), e) maturity level, f) compatibility with customers systems.

It was no difference between the companies size. 7 of 9 respondents indicated that the standards compliance was an important technical issue when evaluating OSS components. Three respondents indicated also “Matching functionality” as an important

issue. *"We need to assure that the component is covering the requirements, that the component was used in other projects. We also look at the design quality, architecture quality, documentation for big components, standards compliance."* indicated one of the respondents.

Therefore we can formulate the next descriptive finding:

DF6: Matching functionality and standards compliance are the most important technical issues when evaluating an OSS component for integration into a system or application.

When asking question 10 ("What organizational issues of the OSS community is your local business considering when selecting an OSS component?"), 8 of 9 respondents answered that the organizational issues of the OSS community are important. Only one company said that they are considering none of the organizational issues of the OSS community, that they usually test only the components.
The most relevant organization issues indicated by the 8 respondents are (in decreasing importance): a) the vitality of the community, b) the quality of end user support, c) the frequency of releases, d) the availability of bug-issues tracking system on project site, e) the road map, f) the time from the last release;

No difference was observed between company sizes.
Most respondents indicated the vitality of the community as one of the most important organizational issues; therefore we can formulate the following finding:

DF7: The vitality is the most important organizational issue of the OSS community when evaluating a component for integration into a system or application.

When asking question 11 ("What professional issues is your local business considering when selecting an OSS component?"), the most relevant answers are (by decreasing importance): a) people select components they have used before, b) people select components written in programming languages they know, c) people select components they have heard about, d) people select components that have been used with success in other OSS components or systems, and e) people select components that are well received in the OSS community.

An important note is that people select components written in programming languages they know only when they need to change the source code, as indicated by one of our respondents: *"The programming language of the component is not necessarily important as long you do not need to change it"*.

Therefore we can formulate the following finding:

DF8: The most important professional issues when evaluating an OSS component are: -People select components they have used already and –People select components

written in programming languages they know when it is necessary to change the source code.

After finding out what companies mean by "good" OSS components, it is also important to discover what companies mean by bad/unsuitable OSS components, i.e., what kind of OSS components companies discard.
When asking question 12 ("What properties make your local business unit discard an OSS component?"), the most relevant findings are (by decreasing importance): a) unsuitable license, b) difficulty to integrate, c) unmatched functionality, d) instability, e) not proved in other projects, f) no fit into the company's development principles, e) not mature enough, f) with bad or no documentation, g) no commercial company supports it, and e) bad performance.
Unmatched functionality is one of the first, if not the first issue which makes a company to discard an OSS component. This issue is mentioned by some of our respondents: *"We try to make sure that a component is adding value for us so the technical requirements are of course important. Also suitable license (compatible with our commercial business) is important. If there is very little information about the component, then we are quite skeptical for using it on a long term."*

We could not observe differences between the company sizes regarding this issue.

DF9: Unmatched functionality, unsuitable license and the difficulty to integrate are the most important properties which may make a company to discard an OSS component.

After finding out the evaluation criteria (described as technical, organization and professional issues), and the properties which make a company to discard an OSS component, the next question is to find out how companies evaluate OSS components.
When asking question 13 ("How does your local business unit evaluate OSS components?"), the most relevant findings are (by decreasing importance): a) do a small test followed by integration, b) perform reviews (e.g. architectural reviews), c) check recommendations, and d) check the popularity.
Companies do not always need to evaluate OSS components. In some cases, companies use known components without being tested before integration: *"If it is a known component, then we just use it. If we do not know the component, we make a prototype and/or try to integrate to see how it works."* In other cases, stable and well-proven components are selected without being tested before integration *"Usually we select components that are stable, so they do not need any prototypes to show that they are good enough."*
In some cases, as indicated by one respondent, only the big components are evaluated: *"If it is a small component with a little impact, the developers can just integrate it. We evaluate only big components."* This statement is related to DF1 and DF2. Not only the software developer takes the final decision about integrating, but also performs the integration. Not only the small components can be selected anytime during the development process, but they can also be integrated without testing. Large components

are usually selected at an early stage during the development process and are also evaluated before integration.

6.7 RQ6: What versions are considered and how to deal with new versions?

This research question is asked by the opened questions 14 to 16 in the interview guide.
A property of an OSS component is its version, therefore is interesting to discover what version companies consider when selecting OSS components.
Asking the question 14 of the interview guide ("Which version does your local business unit usually consider when selecting an OSS component for the first time?"), the most relevant answer is that people select the last (newest) stable version unless it is a good reason (e.g., missing functionality in the stable version) to do otherwise. If this is the case, companies prefer either an older version or a newer version.
"In most cases we select the newest release version of the components. If we find any bugs or problems with it, then we will need to consider an older version", said one respondent from a large company.
No differences were observed between the company sizes regarding this issue. Thus, we can state the following finding:

DF10: Companies consider the last stable version when selecting an OSS component for the first time, unless it is a good reason to consider either a newer or an older version.

When asking question 15 ("Which version do you usually consider when re-selecting an OSS component (for maintaining your system or for a new system)?"), it is important to observe what is the cost and the gain to upgrade. *"For a new system, we go to the latest stable version. For maintaining a system, we stay at the latest stable [version] until we need to fix some security issues or more functionality is needed."* said one of the respondents.

DF11: Companies usually stay at the latest stable version and they only update to a newer version when the gain to update is bigger than the cost to update.

Regarding question 16 ("When does your local business unit update to a new version of the OSS component?"), we have observed some differences between the different company sizes. All the three small companies indicated that they update to a newer version only when the new functionality is significant or when it is necessary to correct bugs. The medium and large companies are concerned about other things in addition, such as to improve security: *"We update to a new version when security is better, new features are needed, or bugs need to be corrected"* said one respondent from a large company.

DF12: Companies update to a newer version of the OSS component when additional functional is needed or when bugs need to be corrected. The medium and the large companies are concerned about other things in addition.

6.8 RQ7: How to maintain the knowledge about the selection processes and the knowledge about the selected OSS components?

The questions 17 to 21 of the interview guide are related to the integration of OSS components and activities after the integration of OSS components.

We want to discover the possible problems after integration, if the selected OSS components have been replaced with other components at some point during the development process.
Asking question 17 ("Has any selected OSS component ever been replaced with a different component at some point during the development process?"), we can observe differences between the company sizes. In large companies, this happens less often than in the medium size companies. In medium size companies, this happens less often than in small size companies. This can probably be correlated to the product size as well, since large companies tend to have larger software products than smaller companies. This issue is, however, a candidate for further studies.

When asking question 18 ("Have you noticed in the end of the development process that your selected component is not suitable?"), 4 of 9 respondents indicated that this has happened during the testing phase. If this happened, *"we first tried to make it works, if not, then to find a replacement, and finally to build ourselves"*, said one of the respondents.

Only one of the large companies indicated that replacement of a selected OSS component with a different component had happened at some point during the development process. Two medium companies indicated that replacement of components had happened both during the implementation phase and during the testing phase.
All the three small companies indicated that replacement of components had happened both during the implementation and testing phase.

DF13: In large companies, replacement of selected OSS companies at some point during the development process happen less often than in medium size companies and in medium size companies this happens less often than in small companies. When this happened, then it was usually done during the development phase and less often in the testing phase. Thus, company size has an effect of the component replacement rate.

When selecting an OSS component, it is always a rationale behind. Very often, several components are evaluated and only one is selected for integration into a system or architecture. Therefore, we want to discover if companies document the rationale behind their choice, how they do this and if they reuse this information later.
When asking question 19 ("Do you document the rationale behind your choice of the selected component?"), half of respondents said they do this, even though not specifically: *"maybe we do not document so much about the choice, but we do document a little bit about what we have introduced into our product like what type of license, the name, and the version of the component."* Other forms of documentation are: a short note, an application architecture document.

Only few respondents said that they reused this information later. The large companies document more the rationale than the medium size companies. The small companies do not document their rationale at all.

DF14: The large companies document the rationale behind the choice of the selected OSS component more than the medium companies. The small companies do not document their rationale.

When asking question 20 of the interview guide ("Do you keep a local knowledge repository about the selected OSS components?"), 8 of 9 respondents indicated that they keep a repository with all the OSS components used in different projects. Can either be a repository with only OSS components or a repository with both COTS and OSS. Some companies do not exactly have a repository with OSS components; it can be "*a repository of projects with the components that are involved and the experiences from using these components.*"
Some respondents said that they also have a kind of knowledge repository about the selected OSS components: a) documents where this information exists, b) a generic knowledge repository, c) internal documents that describe how to use the components, d) a document store, e) a WIKI on the web with information about the selected components. Almost all respondents indicated that they reuse this information later. No differences between the companies size were observed related to this issue.

DF15: Most companies keep a knowledge repository with all the OSS components used in different projects; some companies keep also information about the selected OSS components and they reuse this information later.

When asking question 21 of the interview guide ("Do you have a person who is responsible for the OSS component (e.g. a knowledge keeper)?"), most respondents indicated that they have such a person, even if not always formally. A person can be responsible for all the components (as indicated by one respondent from a small company) or only for some components (as indicated by one respondent from a large company).
Another interesting point is that: "*typically is one or two of the developers. But if it is a very large component, then more people are responsible for it.*"

DF16: Most companies have a person who is responsible for the OSS components, even if this is not always a formal role.

7. Discussion

This chapter will discuss the results from the survey. We will start by presenting the main contributions. Then, a discussion about the validity of the results will follow. Finally, improvements of the survey are suggested.

7.1 Main Contributions

The contributions of this thesis can be divided into four parts: the literature study, new knowledge, a basis for further research, and a reusable research design. After presenting the contributions, we will discuss the results.

7.1.1 Literature study

The first contribution is the literature study. When performing the literature study, the focus was on creating a short introduction to open source by presenting the relevant issues (e.g. "Historical Background of Open Source"). This introduction should help the reader to better understand the rest of the literature study.
The selection of OSS components happens as part of a more general software development process, thus the chapter "Traditional Models, OSS development process and Software Development Process for Composed Based Systems (CBS)" forms a process foundation for the state-off-the-art of selection of OTS components.
The knowledge gained from the literature study has been used to define the research questions. When creating the research questions, we always had in mind to define research questions which could be beneficial for the software industry.

7.1.2 New Knowledge

The most important contribution of this thesis is new or improved knowledge, which has been discovered throughout the study. This knowledge can be divided into: answers to the research questions and descriptive findings.

Answers to the research questions

The answers to the research questions are provided in chapter 6, as qualitative data.

Descriptive findings

We have presented 16 descriptive findings based on the results of the interviews we performed with 9 Norwegian IT companies. These findings cannot be stated as general practice because we interviewed only 9 respondents, but some of the results may be of interest to test further in larger studies. Our descriptive findings are presented below:

DF1: In small companies, it is the software developer who initiates and does the work related to component harvesting, and takes the final decision about integrating OSS components. More roles (in addition to software developer) such as: project manager, project team, architect, system integrator, company leader, are associated to this process in medium and large companies.
DF2: Large components are usually selected at an early stage during the development process, while small components can be selected anytime during the development process.
DF3: Higher quality, shorter time to market, and cost are the principal motivations to

integrate OSS components into a system or application.
DF4: Companies do not use any formal processes for the selection of OSS components.
DF5: Companies usually find OSS components by searching repositories containing components used before, by Web search (Google), specialized web sites, and OSS communities.
DF6: Matching functionality and standards compliance are the most important technical issues when evaluating an OSS component for integration into a system or application.
DF7: The vitality is the most important organizational issue of the OSS community when evaluating a component for integration into a system or application.
DF8: The most important professional issues when evaluating an OSS component are: - People select components they have used already and –People select components written in programming languages they know when it is necessary to change the source code.
DF9: Unmatched functionality, unsuitable license and the difficulty to integrate are the most important properties which may make a company to discard an OSS component.
DF10: Companies consider the last stable version when selecting an OSS component for the first time, unless it is a good reason to consider either a newer or an older version.
DF11: Companies usually stay at the latest stable version and they only update to a newer version when the gain to update is bigger than the cost to update.
DF12: Companies update to a newer version of the OSS component when additional functional is needed or when bugs need to be corrected. The medium and the large companies are concerned about other things in addition.
DF13: In large companies, replacement of selected OSS companies at some point during the development process happen less often than in medium size companies and in medium size companies this happens less often than in small companies. When this happened, then it was usually done during the development phase and less often in the testing phase. Thus, company size has an effect of the component replacement rate.
DF14: The large companies document the rationale behind the choice of the selected OSS component more than the medium companies. The small companies do not document their rationale.
DF15: Most companies keep a knowledge repository with all the OSS components used in different projects; some companies keep also information about the selected OSS components and they reuse this information later.
DF16: Most companies have a person who is responsible for the OSS components, even if this is not always a formal role.

Table 10: Our descriptive findings

7.1.3 A Platform for Future Work

Our work tries to answer as many questions as possible about the selection of OSS components. We have gained new understanding which can be useful for others in further surveys.

7.1.4 Reusable Research Design and Interview Guide

The research design we have developed can be extended and reused. We have provided descriptions of our design and our work, and we have made the interview guide and the help for the interview guide available in Appendix D and E.

7.1.5 Results discussion

Below we compare our results with those from the depth study from autumn 2006, and with those described in literature if any exists, and we relate the research questions to our descriptive findings.

RQ1: Who (which role in a company) initiates and performs the work related to OSS harvesting and when in the development process?

RQ1.1: Who (which role) initiates the OSS harvesting in a company?

RQ1.2: Who (which role) does the work related to the harvesting?

RQ1.3: Who (which role) takes the final decision about integrating an OSS component?

RQ1.4: When in the development process does one select OSS-components?

As indicated in the depth study from the autumn 2006, we did not find a specific description of this in literature and only one descriptive finding was created: "*Usually it is the software architect who initiates the work related to harvesting. The work of harvesting is usually done by the software developer and the final decision about integrating an OTS component is usually done by project manager or software architect.*"

However, in this thesis the first research question is answered by two new descriptive findings, as indicated in Table 11 .
We will use the following notations in the third column of the next tables:

S means supported by the depth study and SL means supported by literature.
PS means partly supported by the depth study and PSL means partly supported by the literature.
NS means not supported in the depth study and NSL means no supported by the literature.
- means not asked in the depth study and –L means not covered by the literature.

RQ1	DF1: In small companies, it is the software developer who initiates and does the work related to component harvesting, and takes the final decision about integrating OSS components. More roles (in addition to software developer) such as: project manager, project team, architect, system integrator, company leader, are associated to this process in medium and large companies.	PS	-L
	DF2: Large components are usually selected at an early stage during the development process, while small components can be selected anytime during the development process.	-	-L

Table 11: RQ1, DF1 and DF2

DF1 from the depth study and DF1 from this master thesis are quite similar. In both cases it is the software developer who does the work of harvesting and the final decision is taken by a project manager. In the depth study, we have not analyzed the results by company size, therefore we have marked "-" for DF2 in Table 11 .
Therefore, our results to Research Question 1 describe existing industry practices about which role initiates the OSS harvesting in when in the development process. These practices need of course to be verified by further and larger surveys.

RQ2: What are the motivations for and experiences of using OSS components?

The second research question is answered in this thesis by DF3, as indicated in Table 12.

RQ2	DF3: Higher quality, shorter time to market, and cost are the principal motivations to integrate OSS components into a system or application.	-	PSL

Table 12: RQ2 and DF3

We did not ask this question in the depth study, therefore we can not compare the answers to this research question with previous results. A survey about the industrial involvement in open source having open source owner, open source participant, inner source participant, and user of open source components as possible company roles, [Hauge and Røsdal2006] indicates that: "High availability of components, source code, and information related to the components are the most important motivations for using OSS components." We explored the role: integrator of open source components, while several roles were explored in [Hauge and Røsdal2006], therefore we think it is natural to obtain different results.
However, further studies having the integrator of open source components as company role need to be performed to verify our results.

RQ3: What is the current process of selecting OSS components?

Based on the answers to this research question, we could only formulate one descriptive finding, as indicated below:

RQ3	DF4: Companies do not use any formal processes for the selection of OSS components.	S	SL

Table 13: RQ3 and DF4

This finding was also proved in the depth study. The existing literature indicates almost the same, that formal selection processes are rarely used [Li2006].
Our result to this research question confirms therefore what is described in literature and the finding from the previous depth study. This finding can be the foundation for future research.

RQ4: How to find OSS components?

Based on the answers to this research question, we formulated a descriptive finding:

RQ4	DF5: Companies usually find OSS components by searching repositories containing components used before, by Web search (Google), specialized web sites, and OSS communities.	PS	PSL

Table 14: RQ4 and DF5

DF5 is a bit similar with an observation from the depth study: "Companies find OTS components by searching on specialized sites or by searching using specialized search engines." The similarities are that companies find OSS components by searching specialized web sites or by using specialized search engines (Google or others). In the literature, it is indicated that: "COTS identification consists of Web searches, product literature surveys and reviews, identification of other reusable system components, and recommendations from external sources" [Morisio et al.2000], which partly support our results.
The results are partly supported by the depth study. In the literature it is described how COTS components can be found, which has some similarities with our results. These results need therefore to be verified by further and larger studies.

RQ5: What are the evaluation criteria when selecting OSS components?

Based on the results to this research question, five descriptive findings were created, as indicated in Table 15.

RQ5	DF6: Matching functionality and standards compliance are the most important technical issues when evaluating an OSS component for integration into a system or application.	S	PSL
	DF7: The vitality is the most important organizational issue of the OSS community when evaluating a component for integration into a system or application.	NS	-L
	DF8: The most important professional issues when evaluating an OSS component are: -People select components they have used already and –People select components written in programming languages they know when it is necessary to change the source code.	S	-L
	DF9: Unmatched functionality, unsuitable license and the difficulty to integrate are the most important properties which may make a company to discard an OSS component.	PS	PSL

Table 15: RQ5, DF6, DF7, DF8, DF9

DF6 is also supported by an observation from the depth study which indicated that: "*The functionality is the most important technical issue when selecting and evaluating OSS*

components. Supports for standards and standards compliance play also an important role". In the depth study we have asked the question giving alternatives and we got the functionality as an important issue to consider. When asking the same question as open question in the master thesis, only few respondents indicated the functionality as an important issue. We believe that the functionality is an important issue for all the respondents anyway, but many of the respondents preferred to refer to other less obvious issues like standard compliance. Therefore, DF6 is partly supported by the depth study. A hypothesis from [Li2006] says that: *"Functional completeness was regarded to be more important than architectural compliance in COTS component selection"*. This shows that functional completeness and architectural compliance are the most important technical issues when selecting COTS. The same issues are indicated by the depth study and the actual results, when selecting OSS. Therefore we believe that is knowledge that companies with the role of integrator of OSS components may be interested in.

DF7 was not supported by the depth study, which indicated the availability of component for test and use as the most important organizational issue of OSS communities. No relevant description was found in literature. Therefore further studies need to be performed to find out what organizational issues of the OSS community are important when selecting components.

DF8 is supported by an observation from the depth study: "The fact that people have experience with the OSS component or that the component is written in programming languages people know, are important issues when selecting and evaluating OSS components". No relevant description was found in literature. Since DF8 was supported by an observation from the depth study, we believe that is knowledge that companies may be interested in.

DF9 is partly supported by a finding from the depth study: *"Unsuitable license and bad code quality are the most important properties which makes a company to discard an OSS component"*. In both cases, unsuitable license was one of the most important properties to discard an OSS component. [Madanmohan and De2004] show that: *"From the perspective of a commercial firm, the license must explicitly permit the distribution of software build from modified source code"*, which confirms that the license should be suitable to the company needs. The depth study and the literature confirm that unsuitable license is a property which makes a company to discard an OSS component. However, further studies need to confirm if also the difficulty to integrate is an important property.

RQ6: **What versions are considered and how to deal with new versions?**

From the answers to this research question, three findings were created, as indicated in Table 16.

RQ6	DF10: Companies consider the last stable version when selecting an OSS component for the first time, unless it is a good reason to consider either a newer or an older version.	S	-L
	DF11: Companies usually stay at the latest stable version and they only	-	-L

	update to a newer version when the gain to update is bigger than the cost to update.		
	DF12: Companies update to a newer version of the OSS component when additional functional is needed or when bugs need to be corrected. The medium and the large companies are concerned about other things in addition.	S	-L

Table 16: RQ6, DF10, DF11 and DF12

DF10 is supported by an observation from the depth study: *"Companies consider the last or the last stable version when selecting and evaluating OSS components."* No relevant description was found in literature. Therefore, the results to this research question describe interesting and new knowledge about what version companies consider when selecting an OSS component for the first time.

We have not asked in the depth study which version companies consider when re-selecting an OSS component (for maintaining a system or for a new system), and no relevant description was found in literature, therefore we could not compare DF11 with anything. Further studies need to verify our finding.

DF12 is also supported by an observation from the depth study: *"Companies update to a new version when additional functionality is needed or when bugs need to be corrected"*. No relevant description was found in literature. Therefore, the results to this research question describe interesting knowledge which can be useful for companies.

RQ7: How to maintain the knowledge about the selection processes and the knowledge about the selected OSS components?

From the answers to this research question, four findings were created, as indicated in Table 17.

RQ7	DF13: In large companies, replacement of selected OSS companies at some point during the development process happen less often than in medium size companies and in medium size companies this happens less often than in small companies. When this happened, then it was usually done during the development phase and less often in the testing phase. Thus, company size has an effect of the component replacement rate.	PS	-L
	DF14: The large companies document the rationale behind the choice of the selected OSS component more than the medium companies. The small companies do not document their rationale.	-	-L
	DF15: Most companies keep a knowledge repository with all the OSS components used in different projects; some companies keep also information about the selected OSS components and they reuse this information later.	NS	PSL
	DF16: Most companies have a person who is responsible for the	S	-L

	OSS components, even if this is not always a formal role.		

Table 17: RQ7, DF13, DF14, DF15 and DF16

DF13 is partly supported by an observation from the depth study: "It may happen that companies realize in the end of the development process that the selected OTS component is not suitable". No relevant description was found in literature. Since in the depth study we were not concerned with comparison across company size and no relevant description was found in literature, further studies need to verify our results.

In the depth study we have not analyzed the data by company size and we did not have an equivalent result for the DF14. We only have the observation: *"Usually people document the decision besides selecting and evaluating OTS components and this is usually done as part of the product/project documentation"*. No relevant description of this was found in literature. Therefore is difficult to take any conclusion, the result should be verified by further studies.

DF15 is not supported by one observation from the depth study: *"Small or medium size companies usually do not keep a knowledge repository about the selected OTS components, only the large companies do"*. However, we could not observe differences between the company sizes related to this issue in the thesis.
This finding is partly supported by [Ruffin and Ebert2004]: "Alcatel maintains a database, which all its engineers can access, containing experiences with 'open source-like' that someone in the company has studied (but not necessary used)." We could not find a specific description with comparison across company size in the literature. Therefore further studies need to verify our result.

DF16 is supported by an observation from the depth study*: "Usually there are people responsible for the OTS components in each company"*. This is new knowledge that can be interesting for companies who integrate OSS components into their systems or applications.

7.2 Validity

Validity is related to how much we can trust the results. [Wohlin et al. 2000] state that adequate validity refers to that the results should be valid for the population of interest. First of all, the results should be valid for the population from which the sample is drawn. Secondly, it may be of interest to generalize the results to a broader population.
The four types of validity described in [Wohlin et al.2000] are: *conclusion*, *internal*, *construct*, and *external validity*.

7.2.1 Conclusion Validity

Conclusion validity is concerned with the relationship between the treatment and the outcome [Wohlin et al.2000].
The first threat is *Low statistical power*. Because we had only 9 respondents, the statistical power of this work is very low. This is a problem we are aware of and therefore

more thorough studies need to be performed to confirm if our results have a more general applicability.
Another threat is *Fishing and error rate.* The researcher may influence the result by looking for a specific outcome. We do not think that we have influenced the results.
Several other threats are presented in [Wohlin et al.2000], but these are not relevant for this study.
The student's lack of experience creating the interview guide and conducting interviews is potentially another problem.

7.2.2 Internal Validity

If a relationship is observed between the treatment and the outcome, we must make sure that is a causal relationship, and that it is not a result of a factor of which we have no control or have not measured [Wohlin et al. 2000].
The relevant threats are:

- Maturation

Subjects may be affected negatively during the interview. In our case, subjects may be tired (some interviews were performed late on the day) and some subjects may not be so motivated to answer.

- Instrumentation

This is the effect caused if the artifacts of the interview are badly designed. We do not think that the interview guide was badly designed and we helped the respondents with additional informational when necessary.

- Selection

Depending on how the subjects are selected from a larger group, the selection effects can vary [Wohlin et al. 2000]. We have used stratified random sampling to select the 9 companies. However, we have used a convenience population, i.e., we have found our respondents not randomly, but we have chosen the ones we knew or were recommended by others.
This may reduce the internal validity of the results. Our group of respondents was only representative for the IT companies, and not for the whole population of companies that develop software (we did not interview non-IT companies with an IT unit) and this may affect the results. It would have been preferable to use the whole population.

7.2.3 Construct Validity

Construct validity concerns generalizing the results of the experiment to the concept or theory behind the experiment [Wohlin et al.2000].
[Jacobsen2005] mentions measuring the right thing as one problem.
Questions and statements in the interview may have been misunderstood or misinterpreted because they were improperly phrased. If the respondents misunderstood something, they may have answered something else.
We realized that some sentences could be better phrased. Since the interview was done by phone, when people had something unclear, we gave them more information.
[Rabson2003] mentions testing and reviews as the best ways of ensuring construct validity. Our supervisors and other researchers have reviewed the interview guide but we did not test the interview guide before the interviews.

7.2.4 External Validity

The external validity is concerned with generalization. If there is a crucial relationship between the construct of the cause, and the effect, the result of the study can be generalized outside the scope of our study? [Wohlin et al.2000]
There are three types of interactions with the treatment: people, place and time. The relevant for us is the people.
Having a subject population not representative for the population we want to generalize to, may affect the result. We had only 9 respondents and they are possible not representative for the larger population.

7.3 Improvements

We think is important to describe ideas of improvement which will be useful both for others when performing further studies:

- Increase the sample size.
- Use the whole population, not only the Norwegian IT companies but also the Norwegian companies with an IT unit.
- Increase the motivation of the respondents.

8. Conclusions and Future Work

8.1 Conclusions

Empirical research is performed to verify theories, develop new theories, or extend the existing ones, and improve practice. This study is mainly used to gain understanding about the selection of OSS components with the ultimate goal to improve the software development practice in industry, particularly to improve the practice of selection of OSS components. This study does can not be used directly to improve the practice of selection and evaluation of OSS components, because further and larger studies needs to be performed in the future to support our results. It is anyway a good step toward the final goal.

We have used the role: integrator of open source, because this is the most appropriate for the research we have performed. More and more companies integrate open source components in their products because the benefits are large. Therefore, improving the practice about the selection of OSS components may help software companies to decrease the time spent. If the time spent is too large, this can offset the advantages of integrating OSS components.

The results of the interview are presented, 16 descriptive findings are formulated based on this. The literature study was very useful to gain understanding about the state-off-the-art but also to define the research questions.

8.2 Future Work

The interviews can be analyzed further to search for more information and knowledge that could be used to build up theories for further studies.

As stated already, this explorative study can be used by others to perform further surveys. It would be very interesting to perform similar but larger surveys in the Norwegian industry but also abroad and then to compare the results by country and by company size. In this way people could share experiences, learn from each other, and improve their processes to select and evaluate OSS components with the ultimate goal to improve development practices.

References

[Alves and Castro2001] Alves, C., Castro, J. "CRE: A Systematic Method for COTS Selection" *Proceedings XV Brazilian Simposium on Software Engineering*, 2001.

[Ayala and Franch2006] Ayala, C. and Franch, X.: "A Goal-Oriented Strategy for Supporting Commercial Off-The-Shelf Components Selection", 9th International Conference on Software Reuse (ICSR 2006). June 11-15, 2006. Torino, Italy.

[Ayala et al.2006] Ayala, C., Sørensen, C.-F., Franch, X., Conradi, R. and Li, J.: Open Source Collaboration for Fostering Off-The-Shelf Components Selection, Accepted at OSS'07 Limerich, Ireland.

[Babbie90] Barbie, E.:Survey Research Methods, Wadsworth, ISBN 0-524-12672-3, 1990.

[Berquist and Ljungberg2001]: The power of gifts: organizing social relationships in open source communities, *Information System Journal* 11, pp.305.320, 2001.

[Boehm88] Boehm, B. W.: A Spiral Model of Software Development and Enhancement, *IEEE Computer*, 21(5): 61-72, May 1998.

[Boehm00] Boehm, B. W.: Requirements that Handle IKIWISI, COTS, and Rapid Change. *IEEE Computer*, 33 (7): 99-102, July 2000.

[Bonaccorsi and Rossi2003] Bonnaccorsi, A. and Rossi, C.: Why Open Source software can succeed, *Research policy*, 2003, vol. 13, no 7, pp. 1243-1258

[Brownsword et al.2000] Brownsword, L., Oberndorf, T., Sledge C.A. "Developing New Processes for COTS-Based Systems" *IEEE Software*, Vol.17, No.;July-August 2000, pp 222-223

[Carney and Long2000] Carney, D. and Long, D.: What Do You Mean bu COTS? Finally, a Useful Answer, *IEEE Computing*, 2000, pp.83-86.

[Chung and Cooper2004] Chung, L., Cooper, K. "Defining Goals in a COTS-Aware Requirements Engineering Approach". *System Engineering*, Vol. 7, No.1, 2004

[Componentsource2007] URL: http://www.componentsource.com

[Cooper and Schindler2001] Cooper, D. R. and P.S. Schindler: Business Research Methods, *McGraw-Hill*. ISDN 0-07-231451-6, 2001.

[COSI project] URL: http://www.idi.ntnu.no/grupper/su/cosi.html.

[Dedrick and West2004] Dedrick, J. and West, J.: An Exploratory Study into Open Source Platform Adoption, *Proceedings of the 37th Hawaii International Conference on System Sciences*, 2004.

[Feller and Fitzgerald2002] Feller, J. and Fitzgerald, B.: Understanding Open Source Software Development, ISBN 0-201-73496-6, 2002.

[Fox et al.1997] Fox, G., Lantner, K. and Marcom, S.: A Software Development Process for COTS-based Information System Infrastructure, *Proc. of IEEE*, pp. 133-142, 1997

[German V-model] German V-model. URL: http://www.v-modell.iabg.de/#AU250, 2005.

[Hauge and Røsdal2006] Hauge, Ø. and Røsdal, A.: A Survey of Industrial Involvement in Open Source, *Master thesis at NTNU*, Department of Computer and Information Science, 2006.

[Hecker200] Hecker, F.: Setting Up Shop: The Business of Open-Source Software, URL http://hecker.org/writings/setting-up-shop.

[Highsmith99] Highsmith, J. A.: Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. *Dorset House Publishing*, New York, USA, 1999.

[Heineman and Council2001] Heinman, G.T. and Council, W.T.: *Component-Based Software Engineering, Putting Pieces Together*. Addison-Wesley, 2001.

[Huang and Yang2005] Huang, M., Yang, .L. and Yang, Y.: A Development Process for Building OSS-Based Applications, SPW 2005, LNCS 3840, pp. 122-135.

[Jacobsen2005] Jacobsen, D. I.: Hvordan Gjennomføre Undersøkelser? Innføring i Samfunnsvitenskaplig Metode (2nd. ed.) Høyskoleforlaget. ISBN 82-7634-663-4, 2005.

[Kent99] Kent, B: Extreme Programming Explained: Embrace Change. *Addison-Wesley*, 1999.

[Kontio96] J. Kontio. A Case Study in Applying a Systematic Method for COTS Selection. *In Proc. of ICSE-18*, pages 201-209, 1996.

[Krogh et al.2003] Krogh, G., Spaeth, S., and Lakhani, K. R.: Community, joining, and specialization in open source software innovation: a case study, *Research Policy*, 2003, pp. 1217-1241.

[Kunda and Brooks1999] Kunda, D., Brooks, L. "Applying Socio-Technical Approach for COTS Selection" *Proceedings UK Acad Inform Syst Conference*, 7-9 April 1999, University of York, pp. 552-565

[Leonard2000] Leonard, A.: Salon Free Software Project, http://www.salon.com/tech/fsp/, Last Accessed June 23, 2001.

[Leung03] Leung, H. K. N. and Leung, K. R. P. H.: Domain-Based COTS-Product Selection Method. Component-Based-Software Quality, LNCS 2693, pp 40-63, 2003

[Li2006] Li, J. "Process Improvement and Risk Management in Off-The-Shelf Component Based Development" *PhD theses at NTNU*, 2006:84, pp 29

[Madanmohan and De2004] Madanmohan, T. R. and De', R.: Open Source Reuse in Commercial Firms, *IEEE Software*, 2004, pp.62-29.

[Maiden and Ncube1998] Maiden, Ncube, C. "Acquiring Requirements for COTS Selection". *IEEE Software* Vol. 15, No. 2, March/April 1998.

[Maiden et al.2002] Maiden, N., Kim, H., Ncube, C. "Rethinking process guidance for Software Component Selection". LNCS 2255, J.C. Dean and A. Gravel (Editors), Springer-Verlang, New York, 2002, pp. 151-164.

[Mockus et al.2002] Mockus, A., Fielding, R.T. and Herbsleb, J.D.: Two Case Studies of Open Source Software Development: Apache and Mozilla, ACM Transactions on Software Engineering and Methodology, Vol 11, No3, July 2002, Pages 309-346.

[Morisio et al.2000] Morisio, M., Seaman, C.B., Parra, A., Basili, V., Kraft, S., and Condon, S. "Investigating and Improving a COTS-Based Software Development Process". *Proceedings International Conference on Software Engineering*, 4-11 June 2000, Limerick, Ireland, pp. 32-41

[Murrain2004] Murrain, M.: Choosing and Using Open Source Software: A primer for Nonprofits. URL: http://www.nosi.net/primer/NOSIPrimer.pdf.

[Navica] The Open Source Maturity Model from Navica.
URL: http://www.navicasoft.com/pages/osmmoverview.htm

[Ochs et al.2001] Ochs, M., Pfahl, D., Chrobok-Diening, G. and Nothhelfer-Kolb, B.: A Method for Efficient Measurement-based COTS Assessment and Selection – Method Description and Evaluation Results, *IEEE Software* 2001, pp. 285-296.

[opensource2006] "The Open Source Definition", http://www.opensource.org/docs/definition.php, 2006

[Robson93] Robson, C.: Experiment, Design and Statistics in Psychology, 3rd edition, Penguin Books, London, England, 1994.

[Robson2003] Robson, C.: Read World Research. Blackwell Publishing. ISBN 0-631-21305-8, 2003.

[Royce87] Royce, W. W.: Managing the Development of Large Software Systems: Concepts and Techniques. *Proc. of the 9th International Conference on Software Engineering*, Mar. 1987, Monterey, California, USA, pp. 328-338.

[Ruffin and Ebert2004] Ruffin, E. and Ebert, C.: Using Open Source Software in Product Development: A Primer, *IEEE Software*, 2004, pp.82-86.

[Scacchi2004] Scacchi, W.: Free and Open Source Development Practices in the Game Community, *IEEE Software*, 2004, pp.59-66.

[Sharp and Robinson2004] Sharp, H. and Robinson, H.: An Ethnographic Study of XP Practice, *Empirical Software Engineering*, 9, 353-375, 2004.

[Strauss and Corbin1998] Strauss, A. and Corbin, J.: Basics of Qualitative Research: Second Edition, *Sage Publications*, ISBN 0-8039-5939-7, 1998.

[Tran et al.1997] Tran, V., Liu, D. B., and Hummel, B.: Component-based Systems Development: Challenges and Lessons Learned. *Proc. of the 8th IEEE International Workshop on Software Technology and Engineering Practice*, Jul. 1997, London, UK, pp. 452-462.

[Vidyasagar and Chang2004] Vidyasagar, P., Chang, E.: Open Source and Closed Source Software Development Methodologies, *Proceedings of the 4th Workshop on Open Source Software Engineering*, May 2004, Edinburg, Scotland.

[Webbink2003] Webbink, M. H.: Understanding Open Source Software. The New South Wales Society for Computers and the Low Journal 51, 1-1. URL: http://www.nswscl.org.au/journal/51/Mark_H_Webbink.html.

[Wheeler2004] Wheeler, D.A.: Generally Recognized as Mature (GRAM) OSS/FS programs. URL http://www.dwheeler.com/gram.html, 2004.

[Wheeler2006] Wheeler, D.A.: How to Evaluate Open Source Software / Free Software (OSS/FS) programs. URL http://www.dwheeler.com/oss_fs_eval.html, 2006.

[Wikipedia.org] URL: http://en.wikipedia.org/wiki/.

[Wohlin et al.2000] Wohlin, C, Runeson, P., Host, Ohlsson, M.C., Regnell, B., Wesslen, A.: Experimentation in Software Engineering, An Introduction, *Kluwer Academic Publishers*, 2000.

[Woods and Guliani2005] Woods D. and G.Guliani: Open source for enterprise: Managing risks, reaping rewards. O'Reilly, 2005., pp 6.

Part 5 - Appendices

Appendix A: Glossary

CBS	Component-based system
Copy-left	Copy-left is a general method for making a program or other work free and requiring all modified and extended versions of the program to be free as well.
COSI	Co-development using inner & Open source in Software Intensive products
COTS	Commercial off the shelf
CVS	Concurrent Versions Systems
GUI	Graphical User Interface
Hacker	A person who is very skilled at computer programming and spends a lot of time programming
IDI	Department of Computer and Information Science
ITEA	Information Technology for European Advancement
NTNU	Norwegian University of Science and Technology
OTS	Off-the-Shelf
OSS	Open Source Software
Qualitative	Concerned with information as text
Quantitative	Concerned with information as numbers

Appendix B: The Open Source Definition

Introduction

Open source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code

The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost preferably, downloading via the Internet without charge. 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 preprocessor 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" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

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. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the

program is redistributed should have the same rights as those that are granted in conjunction with the original 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. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

*10. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.

Appendix C: OSS Development Process

The process model in Open source software development (OSSD) is different from Closed Source Software Development (CSSD) [Vidyasagar and Chang2004]. CSSD normally follows a spiral or iterative model of development, i.e software development goes through all phases like planning, design, and implementation, whereas OSSD follows an evolutionary model for development where the software never reaches a final state and keeps on evolving. OSSD is more a concurrent or parallel process.

There is, of course, no single OSS development process. However, [Feller and Fitzgerald2002], point out a number of characteristics that are found in the majority of the OSS projects:

- The OSS process generally involves *large, globally distributed communities of developers collaborating primarily through the Internet*;
- The OSS developers work in *parallel*;
- OSS development communities often exploit the power of *peer review* to facilitate the debugging process;
- OSS projects are generally characterized by *rapid, incremental release schedule*, as opposite to the proprietary software;
- OSS has attracted a very large pool of *experienced and esteemed developers*;
- OSS projects are coordinated by *highly motivated communities*;

Despite the many differences from proprietary software, it is important to note that OSS usually goes through the same stages as a proprietary product [Murrain2004].
Some key differences between proprietary software and OSS are:

- In open source project development, this process may happen much more organically - starting from a single developer to more developers who gradually become interested in the project;
- The pace of open source development can be slower, due to the voluntary nature;
- The quality of OSS can be much better than proprietary software (because of many adopters and testers);
- The version numbering of open source software tends to be more conservative.

According to its proponents, open source style software development has the capacity to compete successfully, and perhaps in many cases displace, traditional commercial development methods [Mockus et al.2002]. The most often mentioned differences between the open source style development and the traditional commercial development methods are:

- OSS systems are built by potentially large numbers of volunteers. However, currently a number of OSS projects are supported by companies and some participants are not volunteers;
- Work is not assigned; people undertake the work they choose to undertake;
- There is no explicit system-level design, or even detailed design;

There is no project plan, schedule, or list of deliverables.

Appendix D: Interview guide

Spring 2007

Selection of Open Source Components – A Qualitative Survey in Norwegian IT Industry

Marinela Gerea

Thank you for deciding to participate in this survey about selection of Open Source Components (OSS).

Some important definitions we are using in the survey are:

OTS	Off-The-Shelf
COTS	Commercial-Off-The-Shelf
OSS	Open Source Software
In-House-Build	Components build by your company.

We are interested to discover the selection process of OSS components which are going to be integrated into a system or application. We are not interested in infrastructure OSS (e.g. Linux) or open source tools (e.g. Eclipse, MySQL).

Please refer to the general process of selection of OSS components which is used into your local business unit.

Part1. Who initiates the OSS harvesting and when in the development process?

1. Who initiates the work related to OSS harvesting in your local business unit?

_____

2. Who does the work related to OSS harvesting in your local business unit?

_____

3. Who takes the final decision about integrating an OSS component?

_____

4. When in the development process does your local business unit normally select OSS-components?
Are there any differences between large, medium and small components?

_____

_____

Part 2. Motivations for using OSS components

5. What are your local business unit motivations for integrating OSS components into your systems/applications?

_____

_____

Part 3. Selection process

6. Does your local business unit use any formal process for selection of OSS components?

- No
- Yes

If yes, which method and how did you find that method?

_____

7. What is your selection process of OSS?

Please describe the main activities:

_____

_____

_____

Part 4. Finding the OSS components

8. How do you find OSS components?

_____

_____

Part 5. Evaluation criteria

9. What technical issues is your local business unit considering when selecting an OSS component?

_____

_____

10. What organizational issues of the OSS community is your local business considering when selecting an OSS component?

_____

_____

11. What professional issues is your local business considering when selecting an OSS component?

_____

_____

12. What properties make your local business unit discard an OSS component?

_____

_____

13. How does your local business unit evaluate OSS components?

_____

_____

Part.6 Version issues in OSS

14. Which version does your local business unit usually consider when selecting an OSS component for the first time?

_____

15. Which version do you usually consider when re-selecting an OSS component (for maintaining your system or for a new system)?

_____

16. When does you local business unit update to a new version of the OSS component?

_____

Part 7. Integration of OSS components

17. Has any selected OSS component ever been replaced with a different component at some point during the development process?

- No
- Yes

If yes, in which phase? _____

18. Have you noticed in the end of the development process that your selected component is not suitable?

- No
- Yes

If yes:

In which phase?

_____

Did you start building an internal component instead of searching for an OSS component?

Part 8. Activities after the integration of OSS components

19. Do you document the rationale behind your choice of the selected component?

□ Yes
□ No
If yes, please specify how:_____

_____
If yes, do you reuse this information later? □ Yes
□ No

20. Do you keep a local knowledge repository about the selected OSS components?

□ Yes
□ No
If yes, please describe shortly : _____

_____
If yes, do you reuse this repository later? □ Yes
□ No

21. Do you have a person who is responsible for the OSS-component (e.g. a knowledge keeper)?

□ No
□ Yes, we have such a person in our local business unit

Part 9. About the company

1. What is the name of your company or business unit?

Company name :_____

2. What is the type of company?

_____

3. What is the ownership of your company?

_____

4. What is the staff size of your mother company in your own country? (full- & part-time persons)_____?

5. What is the staff size of your local business unit (full- & part-time persons)_____?

[This staff size may be the same for small and medium sized companies]

6. How many software developers are working in your local business unit? _____

7. What is the main business area of your company?

_____

8. What type of OSS does your company already use?

- ☐ LAMP (Linux, Apache, MySQL, PHP)
- ☐ Desktop applications (Evolution, FireFox, OpenOffice, etc)
- ☐ Development tools
  - ☐ Bugzilla
  - ☐ CVS
  - ☐ Subversion
  - ☐ Emacs
  - ☐ GNU Compiler
  - ☐ Other: _____

- Languages
  - Perl
  - PHP
  - Python
  - Java
  - Other: _____
- Web Application Development/Content Management (Zope, Plone, Midgard, eZ Publish, etc)
- Graphical user interface (GNOME, KDE, XFree86, etc)
- Security (Nessus, Nmap, Open SSH, Open SSL, Snort, etc)
- Research tools (Octave, R, etc)
- Other (please specify):_____

9. How would you describe your software process model?

- Code and fix (writing code as long there is time and money, without a set of requirements)
- Waterfall (a sequential, document-driven methodology)
- V-model (extended waterfall model, adding details on the validation and verification phase)
- Spiral (development into small pieces)
- Evolutionary Prototyping (the model starts with an initial idea, which is then prototyped and released to the customer)
- Unified Process (risk- and use-driven, architecture-centric, iterative and incremental software development process)
- Agile models
  - Adaptive Software Development (ASD) (iterative, risk- and mission-driven, component based, time-boxed process)
  - XP (flexible, lightweight, people- and result-oriented development process that allows for requirements changes at any time during development)
  - Scrum (trying to have everybody involved, the requirements from customers are priorities, daily meetings with the customers)
- Other (please specify):_____

10. How many selection processes do your local business unit performs by year?

11. How much effort (in person hours) does it take to perform a selection process?

____ person hours.

Part 10. About the respondent

1. How old are you?

- □ <25
- □ 26-30
- □ 31-35
- □ 36-40
- □ 41-50
- □ 51-60
- □ >60

2. What is your current position in your business unit?

- □ IT Manager
- □ Project manager
- □ Software architect
- □ Software developer
- □ Web designer
- □ Tester/Quality assurance
- □ Other :_____

3. How long have you been working in the current business unit?

Years :_____

4. How long have you been working with software development?

Years :_____

5. How long have you been working with OSS?

Years :_____

6. How many projects using OSS components have you been involved in?

Number :_____

7. What is your highest completed education?

- □ Bachelor (BSc)
- □ Master (MSc)

- Ph.D.
- Other education : _____

8. Is your degree software-related (informatics/computer science/telecommunications)?

[Mark one alternative]

- Yes
- No

Appendix E: Help for the interview guide

Spring 2007
Selection of Open Source Components – A Qualitative Survey in Norwegian IT Industry

Marinela Gerea

Help file

Thank you for deciding to participate in this survey about selection of Open Source Components (OSS).

Some important definitions we are using in the survey are:

OTS	Off-The-Shelf
COTS	Commercial-Off-The-Shelf
OSS	Open Source Software
In-House-Build	Components build by your company.

We are interested to discover the selection process of OSS components which are going to be integrated into a system or application. We are not interested in infrastructure OSS (e.g. Linux) or open source tools (e.g. Eclipse, MySQL).

Please refer to the general process of selection of OSS components which is used into your local business unit.

Part1. Who initiates the OSS harvesting and when in the development process?

1. Who initiates the work related to OSS harvesting in your local business unit?

Please select one or several alternatives.

- IT Manager
- Project manager
- Software architect
- Software developer
- Other (please specify):_____

2. Who does the work related to OSS harvesting in your local business unit?

Please select one or several alternatives.

- IT Manager
- Project manager
- Software architect
- Software developer
- Other (please specify):_____

3. Who takes the final decision about integrating an OSS component?

Please select one or several alternatives.

- IT manager
- Project manager
- Software architect
- Software developer
- Tester/Quality assurance
- Customer
- Other (please specify):_____

4. When in the development process does your local business unit normally select OSS-components?

Are there any differences between large, medium and small components?

a) For large components

Pre-study
Requirements/Analysis

Overall design
Detailed design
Coding
Incrementally throughout the project

b) For small and medium size components

Pre-study
Requirements/Analysis
Overall design
Detailed design
Coding
Incrementally throughout the project

Part 2. Motivations for using OSS components

5. What are your local business unit motivations for integrating OSS components into your systems/applications?

Licenses are cheaper
Installation cost is smaller
Maintenance cost with OSS is smaller (than with other software)
Learning cost is smaller
To be more independent from the big companies' prices, support.
Want to support the OSS movement
It is recommended by our partners/customer
Our developers work on open source projects in their spare time
We want to increase our skills in this area
The market is looking for this
Compliance to standards
Idealism
OSS components are easily available for test and use

Part 3. Selection process

6. Does your local business unit use any formal process for selection and evaluation of OSS components?

- No, our process is based on the intuition approach
- Yes

If yes, which method and how did you find that method?

_____

7. What is your selection process of OSS?

Please describe the main activities (searching, screening, evaluation, integration e.g):

_____
_____
_____
_____

Part 4. Finding the OSS components

8. How do you find OSS components?

Please select one or several:

- ☐ Get information/the component from a colleague/friend
- ☐ Look at "generally recognized as mature" OSS programs (i.e. http://www.dwheeler.com/gram.html)
- ☐ Search on specialized sites (please indicate which sites):_____ _____
- ☐ Search using specialized search engines (i.e. Google's specialized searches for Linux)
- ☐ Read articles, books or magazines
- ☐ Use your local business unit internal "knowledge base"
- ☐ Advice from forums or mailing lists
- ☐ Other (please specify):_____

Part 5. Evaluation criteria

9. What technical issues is your local business unit considering when selecting an OSS component?

Matching functionality
Extra functionality
Code quality
Design quality

Architecture quality
Documentation quality
Security
Performance
Integration with other software
Have standard-based API
Standards compliance
Programming language/environment

10. What organizational issues of the OSS community is your local business considering when selecting an OSS component?

Vitality of community
Quality of end-user support
Frequency of releases
Quality of project site
Availability of roadmap/plan
The availability of component for test and use
Availability of bug-issues tracking system on the project site

By Vitality of community we mean: Activity on mailing lists/email, Number of downloads, Number of page views, Time to fix bugs, Time to implement new functionality.

11. What professional issues is your local business considering when selecting an OSS component?

People select components they already have used
People select components they have heard about
People select components written in programming languages they know
Evaluation time

12. What properties make your local business unit discard an OSS component?

Is ill-suited for the technical skill set of people responsible for integrating and maintaining it.
Little community activity
Does not have momentum
Has unsuitable license
With bad or no documentation
No commercial company supports it

13. How does your local business unit evaluate OSS components?

Create prototype software and a temporary integration and testing of the candidate
Use an evaluation process which is well documented in your local business unit
Use a document checklist to evaluate the OSS components
Performe testing and/or prototyping with OSS components
Performe code reviews of OSS components
Performe architecture reviews of OSS components

Part.6 Versions issues OSS

14. Which version does your local business unit usually consider when selecting an OSS component for the first time?

Please select one:
- The last stable version
- The version that most people use
- The version that provides more functionality
- Other (please specify):_____

15. Which version do you usually consider when re-selecting an OSS component (for maintaining your system or for a new system)?

Please select one:
- The last stable version
- The version that most people use
- The version that provides more functionality
- Other (please specify):_____

16. When does you local business unit update to a new version of the OSS component?

Please select one or several:
- Each time you release a new version of the system containing one or more OSS components
- Each time a new version of the OSS component is coming
- When additional functionality is needed
- When bugs need to be corrected
- Other (please specify):_____

Part 7. Integration of OSS components

17. Has any selected OSS component ever been replaced with a different component at some point during the development process?

□ No
□ Yes
If yes, in which phase? _____

18. Have you noticed in the end of the development process that your selected component is not suitable?

□ No
□ Yes
If yes:

In which phase?

_____

Did you start building an internal component instead of searching for an OSS component?

_____

Part 8. Activities after the integration of OSS components

19. Do you document the rationale behind your choice of the selected component?

□ Yes
□ No
If yes, please specify how:_____

_____
If yes, do you reuse this information later? □ Yes
□ No

20. Do you keep a local knowledge repository about the selected OSS components?

□ Yes
□ No
If yes, please describe shortly : _____

_____
If yes, do you reuse this repository later? □ Yes
□ No

21. Do you have a person who is responsible for the OSS-component (e.g. a knowledge keeper)?

□ No
□ Yes, we have such a person in our local business unit

Part 9. About the company

1. What is the name of your company or business unit?

Company name :_____

2. What is the type of company?

[Mark one alternative]
- □ Stand-alone: The highest reporting entity with no parent organization above it
- □ Subsidiary: An independent entity with majority interest held by a parent.

3. What is the ownership of your company?

[Mark one alternative]
- □ Publicly traded on a stock exchange
- □ Privately held company
- □ Government, education, or non-profit organization

4. What is the staff size of your mother company in your own country? (full- & part-time persons)_____?

5. What is the staff size of your local business unit(full- & part-time persons)_____?

[This staff size may be the same for small and medium sized companies]

6. How many software developers are working in your local business unit? _____

7. What is the main business area of your company?

[Mark one alternative]
- □ IT/Telecom industry (Ericsson, Nokia, NERA, etc)
- □ Telecom service provider (Telenor, Telefonica, Orange, etc)
- □ Software / IT consulting company (Accenture, CapGemini E&Y, TiedoEnator, etc)
- □ Retail product with large degree of embedded software (Philips, Sony, etc)
- □ Software health care
- □ Research
- □ Oil company/service company
- □ Other software-intensive company (please specify) :_____

8. What type of OSS does your company already use?

- LAMP (Linux, Apache, MySQL, PHP)
- Desktop applications (Evolution, FireFox, OpenOffice, etc)
- Development tools
  - Bugzilla
  - CVS
  - Subversion
  - Emacs
  - GNU Compiler
  - Other: _____
- Languages
  - Perl
  - PHP
  - Python
  - Java
  - Other: _____
- Web Application Development/Content Management (Zope, Plone, Midgard, eZ Publish, etc)
- Graphical user interface (GNOME, KDE, XFree86, etc)
- Security (Nessus, Nmap, Open SSH, Open SSL, Snort, etc)
- Research tools (Octave, R, etc)
- Other (please specify):_____

9. How would you describe your software process model?

- Code and fix (writing code as long there is time and money, without a set of requirements)
- Waterfall (a sequential, document-driven methodology)
- V-model (extended waterfall model, adding details on the validation and verification phase)
- Spiral (development into small pieces)
- Evolutionary Prototyping (the model starts with an initial idea, which is then prototyped and released to the customer)
- Unified Process (risk- and use-driven, architecture-centric, iterative and incremental software development process)
- Agile models
  - Adaptive Software Development (ASD) (iterative, risk- and mission-driven, component based, time-boxed process)
  - XP (flexible, lightweight, people- and result-oriented development process that allows for requirements changes at any time during development)
  - Scrum (trying to have everybody involved, the requirements from customers are priorities, daily meetings with the customers)
- Other (please specify):_____

10. How many selection processes do your local business unit performs by year?

____ processes.

11. How much effort (in person hours) does it take to perform a selection process?

____ person hours.

Part 10. About the respondent

1. How old are you?

- <25
- 26-30
- 31-35
- 36-40
- 41-50
- 51-60
- >60

2. What is your current position in your business unit?

- IT Manager
- Project manager
- Software architect
- Software developer
- Web designer
- Tester/Quality assurance
- Other :_____

3. How long have you been working in the current business unit?

Years :_____

4. How long have you been working with software development?

Years :_____

5. How long have you been working with OSS?

Years :_____

6. How many projects using OSS components have you been involved in?

Number :_____

7. What is your highest completed education?

- Bachelor (BSc)
- Master (MSc)
- Ph.D.
- Other education : _____

8. Is your degree software-related (informatics/computer science/telecommunications)?

[Mark one alternative]

- Yes
- No