An empirical study on Open Source Selection Strategies in the Norwegian software industry

TDT4520 - Program and Information Systems, Specialization Project

A project assignment by Ketil Sandanger Velle at

Norwegian University of Science and Technology (NTNU)

Department of Computer and Information Science

Autumn 2008

Main Supervisor: Reidar Conradi[1]

Co-advisor: Øyvind Hauge[1]

Contributors: Claudia Ayala[2]*, Xavier Franch*[2] *and Jingyue Li*[1]

[1] Norwegian University of Science and Technology
[2] Universitat Politécnica de Catalunya

Abstract

Of-The-Shelf (OTS) components are in use in many development projects today (Basili, 2001) and may prove to give significant advantages in form of producing software in shorter time and with better quality than what could have been done without such components. The OTS components could be of different types, e.g. Commercial-Of-The-Shelf (COTS) components or Open Source Software (OSS). This report is mainly focused on OSS components.

To get a grip on the size of the OSS community one could look at the number of ongoing OSS projects. At the end of 2006 there were over 100 000 ongoing OSS projects in the major repositories like SourceForge, CodeHaus, Tigris, Java.net and Open Symphony (Taibi, Lavazza, & Morasca, 2007). Additionally OSS has a large number of big success stories, e.g. Apache, Mozilla, Eclipse and MySQL, which is used by millions of users worldwide. As the amount of available OSS components is constantly increasing the selection of components gets harder. Choosing the wrong component could have large or even disastrous effects for a project. Therefore a wide range of research has been performed on the selection of components, but this research has mainly been focused on formal methods.

Little empirical data exist on how software companies approach selecting OSS components. The only thing that is clear is that the formal methods, proposed in research literature, do not seem to be in use. In order to improve the quality and reliability of the software industry's choice of technologies we first have to get a picture of what is done today.

Therefore the questions that are investigated in this study are (1) how do companies identify possible OSS components? After identifying them, (2) how are they evaluated? Concluding the selection process, (3) how is the final choice done? In addition to this it was interesting to investigate (4) which resources that are used to aid this selection process?

This report is a result of an empirical study in Norwegian software industry concerning how companies approach selecting OSS components for their development projects. Information was gathered through semi-structured interviews at volunteering companies in Norway.

The findings of the research are that the identification is a twofold process consisting of a proactive and a reactive part. The evaluation phase seems to be of a more ad-hoc manner than following the formal methods presented in literature, and the phase does not seem to be very strict in regards to evaluation criteria and standard activities. Instead companies put much emphasis on downloading the component and creating a simple prototype. Further the final choice is often done in a first-fit manner and internet resources and social resources are important to aid this whole selection process.

Preface

The report is part of the course TDT4520 - Program and Information Systems, Specialization Project and was produced during the autumn of 2008. This project work's objective is to create a foundation for the master thesis in the following semester. The foundation is created by using scientific methods, acquiring knowledge from literature and working independently under the guidance of a supervisor.

I would like to thank the contributors to my project Claudia Ayala, Xavier Franch and Jingyue Li for their work and comments on the interview guide, and my supervisors Reidar Conradi and Øyvind Hauge for guidance throughout the project. A special thanks to Øyvind Hauge for helpful comments and advice during weekly meetings, over e-mail and through online direct communication. Finally I would like to thank the five participating companies and respondents for their voluntary participation in my research.

Trondheim 17.12.2008

_____

Ketil Sandanger Velle

Table of contents

Part 1 Introduction 1

1 Introduction 1

Part 2 Pre-Study 3

2 Introduction to Of-The-Shelf components 3

2.1 COTS 3

2.2 OSS and Free Software 3

2.3 OSS Licensing 4

2.4 Advantages and drawbacks with using OSS components in software engineering 5

3 Selection of OTS components 7

3.1 Identification 7

3.2 Evaluation 9

3.3 Formal OSS evaluation methods 10

3.4 Choice 11

4 Challenges in OSS development 12

Part 3 Research 15

5 Research design and context 15

5.1 Motivation 15

5.2 Research question(s) 15

6 Research method 17

6.1 Sampling 18

6.2 Analysis of data 19

Part 4 Results, Discussion and Conclusions 21

7 Results 21

7.1 RQ1: Identification 21

7.2 RQ2: Evaluation 22

7.3 RQ3: Choice 24

7.4 RQ4: Resources 25

7.5 Other findings 27

8 Discussion 30

8.1 The selection process 30
8.2 Formal methods not used 31
8.3 Lack of an solution to aid the selection process 31
8.4 Poor OSS documentation 32
9 Limitation and validity 34
10 Conclusions and Future Work 35

List of tables

Table 1: Identification methods 8
Table 2: Evaluation Criteria 9
Table 3: Key figures Norwegian ICT-sector 15
Table 4: Mapping between questions in interview guide and research questions 18
Table 5: Company information 19
Table 6: Respondent information 19
Table 7: The twofold identification process 21
Table 8: Evaluation criteria used by the companies 23
Table 9: Internet resources used in the identification of OSS components 25
Table 10: Quotes considering the usefulness of Internet resources 26
Table 11: OSS technologies used by the different companies 27
Table 12: Quotes concerning lack of documentation in OSS projects 28
Table 13 High level goals of the OTS-Wiki 31
Table 14: Appendix - OSS component information VII
Table 15: Appendix - URL for transcripts IX

Part 1 Introduction

1 Introduction

Of-The-Shelf (OTS) components are in use in many development projects today (Basili, 2001) and may prove to give significant advantages in form of producing software in shorter time and with better quality than what could have been done without such components. The OTS components could be of different types, e.g. Commercial-Of-The-Shelf (COTS) components or Open Source Software (OSS). This report is mainly focused on OSS components.

To get a grip on the size of the OSS community one could look at the number of ongoing OSS projects. At the end of 2006 there were over 100 000 ongoing OSS projects in the major repositories like SourceForge, CodeHaus, Tigris, Java.net and Open Symphony (Taibi, Lavazza, & Morasca, 2007). Additionally OSS has a large number of big success stories, e.g. Apache, Mozilla, Eclipse and MySQL, which is used by millions of users worldwide. As the amount of available OSS components is constantly increasing the selection of components gets harder. Choosing the wrong component could have large or even disastrous effects for a project. Therefore a wide range of research has been performed on the selection of components, but this research has mainly been focused on formal methods.

Little empirical data exist on how software companies approach selecting OSS components. The only thing that is clear is that the formal methods, proposed in research literature, do not seem to be in use. In order to improve the quality and reliability of the software industry's choice of technologies we first have to get a picture of what is done today.

Therefore the following research questions were formed:

- *RQ1: How do Norwegian software companies identify possible OSS components?*
- *RQ2: How are the identified OSS components evaluated?*
- *RQ3: How is the final choice of the OSS components accomplished?*
- *RQ4: Which resources do Norwegian software companies use or to aid the OSS selection process?*

The report is a result of an empirical study in Norwegian software industry concerning how companies approach selecting OSS components for their development projects. Data was gathered through semi-structured interviews at volunteering companies in Norway. In parallel with these interviews in Norway, interviews were also conducted in Spanish software industry. A total of five interviews were conducted with developers in the Norwegian Software Industry. The participating companies was found either as a

result of my own knowledge of the industry or as a result of a search on the yellow pages[1].

My contribution is some empirical input on how the whole selection process is carried out by companies in the Norwegian software industry. Starting with the identification, continuing with the evaluation and ending with a final choice interviewees were asked to elaborate how this was done in a project in which they had participated. This data was analyzed to find similarities which were generalized into concepts which are presented in the report. In addition some data on which resources companies use to aid the selection process is presented.

The remainder of this report is in three more parts. Firstly Part 2 presents a pre-study comprising of an introduction to OTS components, information on the selection of OTS components and finally a section about challenges connected to the use of the components. The next part is a research part where the motivation for the research and the research question is introduced. In addition the used research method is presented, together with information about sampling and how to analyze the data gathered through the interviews. The fourth and last part of the report present the results gathered through this research and how they answer the research questions. After that follows a discussion of the results, together with an insight on limitation and validity of this study. The report ends with a conclusion and some suggestions for future work.

[1] http://www.gulesider.no

Part 2 Pre-Study

2 Introduction to Of-The-Shelf components

Of The Shelf (OTS) components are reusable software components produced by a third party. These components can be of different types, i.e. Commercial-Off-The-Shelf (COTS) components or Open Source Software (OSS) (Li, Conradi, Bunse, Marco, Slyngstad, & Morisio, 2008). OTS components are finished pieces of code that could ease the development of a software project. Using OTS components is part of what is called Component-based software development (CBSD) which has its advantages in that not everything has to be coded from scratch. This in turn could lead to a more effective development process, shorten the development time and reduce the development overall costs.

OTS components could be tailored (customized) to fit requirements of the application that is being developed. Such tailoring could be done by e.g. using add-ons or adjusting parameters or modifying the source code if you have that available.

2.1 COTS

COTS components are pieces of software that could be bought and used as a building block to develop new applications (Vigder, Gentleman, & Dean, 1996). These components are available for a fee, and oftentimes licensing costs are bundled with the components as well.

As the acronym implies the component is commercial and consequently you have a vendor/firm that sold it to you. The vendor has to market the product in such a way that interested companies will buy the software component. It may be that the vendor additionally could provide support and updates to the component the adaptor bought, and maybe even assist the adaptor in integrating the component in his project or system.

The source code of the COTS component is usually not available for the adaptor; therefore it is most often not likely that the adaptor could customize the code of the COTS component to tailor the application to his needs. However in many cases the component has some kinds of parameters that could be tweaked to better fit the adaptor's need, but these could be quite limited. In the case this is not enough, the vendor may be willing to support the adaptor solve the problem.

2.2 OSS and Free Software

In contrast to the COTS components OSS does not necessarily have a commercial vendor/firm as a provider but rather the provider is a community of developers. Most importantly OSS components are free and do not include any licensing costs or one time fees. Therefore OSS has become a cost-efficient option for quality software development (Merilinna & Matinlassi, 2006). OSS is often interchanged with Free Software (FS) which has many similarities but is not completely identical (Free Software Foundation, Inc.). Primarily the difference lies in ideological issues, as is summarized in

this quote from the GNU website[1]: *"For the Open Source movement, non-free software is a suboptimal solution. For the Free Software movement, non-free software is a social problem and free software is the solution."* FS could be defined as software (components) with freely available source code which could be altered to fit the developer's specific need. FS has a philosophy consisting of four freedoms (Free Software Foundation, Inc.):

1. The freedom to run the program for any purpose.
2. The freedom to study how the program works and change it to fit individual needs.
3. The freedom to redistribute the program to help others.
4. The freedom to improve the program and release a new version so that the whole community benefits from the changes made.

A precondition for both point 2 and point 4 is that the source code is available for download.

As mentioned the OSS definition has a slightly different definition (Open Source Initiative, 2006) than FS but they are practically very similar and their differences have no real influence on this report. Therefore for the remainder of this report the term OSS will be used as a common term for both Free Software, Open Source Software and any other terms with virtual the same meaning(e.g. Free and Open Source Software(FOSS), Free/Libre/Open Source Software (FLOSS) etc.)

It is also important to note that OSS is not the same as Freeware. Freeware are distributed without a fee but also without source code access. In addition, freeware often have a certain trial period and when that period expires a fee has to be paid if the user wishes to keep using the software.

2.3 OSS Licensing

A copyright protects the intellectual property of the creator(s) and gives exclusive rights to control the distribution of his/their work. An OSS license grants adaptors with the four freedoms mentioned above, but may set restrictions when distributing proprietary software in which the component is used. In other words an OSS license is a legal agreement which defines how the licensed component could be used.

An enormous amount of different licenses have emerged during the last years and in 2006 almost 100 distinct licenses where approved by either the Open Source Initiative (OSI) or the Free Software Foundation (FSF) (Fitzgerald, 2006). Three of the most well-known and used licenses is the GPL license, the L-GPL license and the BSD license. The GPL license is very restrictive when releasing non-free software with a GPL signed component. It simply states that if a GPL component is used in a program, the program itself has to be released under the GPL license. It is the most widely used license, Scacchi reports that almost half of the components at SourceForge were licensed under GPL as

[1] http://www.gnu.org/philosophy/free-software-for-freedom.html

of July 2007 (Scacchi, 2007). The L-GPL license is a bit lesser restrictive since it only demands that any modification of the component has to be released. In contrast the BSD license is much more liberal and you could without any problem release it with proprietary software linking (Rusin).

The licenses could be divided into two main groups; Viral or non-viral. The word viral has been adjudged to have negative connotations (Fitzgerald, 2006), therefore the term reciprocal is used instead. GPL is an example of a reciprocal license, BSD is a non-reciprocal license. When considering an OSS component the adaptor must also carefully assess what license the component has and which consequences this license has for the software that is being developed. Recently FSF has filed a lawsuit against Cisco since license agreements of several components (including the GCC, binutils and the GNU C library) used by Cisco are not followed (Free Software Foundation, 2008). Most of these are components licensed under GPL but Cisco has failed to honor the legal agreement by not releasing their source code with the distributed software. This is a clear example of how important it is to fully understand the license and its consequences.

Copyleft has become a well known term to describe how a license ensures that the licensed software has the same freedoms in the modified versions as it had in the original version. GPL is the most widely known copyleft license.

2.4 Advantages and drawbacks with using OSS components in software engineering

The four freedoms ease the CBSD since the component is more flexible in terms of tailoring and customization to the firm's specific needs due to that the source code is freely available to see, modify and re-release. The fact that the code is available eases the assessment of the internal quality of the component, which is mainly done by looking at code metrics (Taibi, Lavazza, & Morasca, 2007). Many of these metrics are effectively supported by tools such as Chidamber and Kemerer's object-oriented metrics suite (Aivosto Oy), Halstead complexity metrics (Testwell Oy , 2007) etc. Having access to modify the code could however cause an increase in maintenance costs. Further as the first freedom states, the component could be used in any kind of software and for any purpose the development firm has in mind. The only thing that may restrict this is the license of the OSS component, which was discussed in the previous subsection.

OSS components are highly available and are therefore easy to test and adopt. Many of these components have proven to be of good quality. This may be due to that many OSS components often have large communities which develop, test, report bugs and releases fixes and updates in a rapid and good manner. The component is therefore thoroughly tested by different people and maybe in different production environments. This is summarized in a famous quote which Eric S. Raymond named Linus's Law (Raymond, 2001):

"Given enough eyeballs, all bugs are shallow."

An additional reason why this has proven to be a fact is that the developers create and test code directly on how the software is to be used. This avoids the problem of feature creep that commercial products tend to have. Feature creep is *"…focusing on delivering more features in a race to outshine competitors rather than on what product users really need."*(Golden, Succeeding with Open Source, 2005). By releasing early and often, which is a normal practice in the OSS community the users use (and misuse) the software giving feedback to the developers in a quick and effective manner, and maybe even in a greater scale than what could have been the case without an OSS solution.

Another major benefits using OSS software is unrestricted amount of installations. Therefore the usual dilemma of wanting more installations than the company could afford could be avoided, which may be the case with COTS components.

A notable issue when using OSS components is that of when to update. Typically in a software project using OSS, several OTS components are used and often glue-code has been produced to get the components to work together as a unity. Then when an OSS product has a new update, the development team has to carefully consider if this update is worthwhile, since it may cause much overhead to update the glue code and modifications of the already integrated OSS component. On the other hand update may be fixing a serious bug or present new and advantageous functionality so it may not always be an easy choice.

Ari Jaaksi shows an example of development with OSS components in his article from 2007 (Jaaksi, 2007). He presents the telephone vendor Nokia's experience with developing software using OSS components. The development of two of their Internet Tablets mobile consumer devices used approximately 75% OSS components, 25% of which were used without any modification. Nokia's experience in OSS integration is with few exceptions good. They were able to create product in shorter time and with fewer resources than they would have been able to do with own implementation efforts. Also the quality of the OSS software is often better than the quality of the software they developed themselves. This is mainly because the OSS software has been used, tested and debugged before Nokia take it into use and the most severe errors and bugs has already been found and fixed.

As seen with Nokia experience, development using OSS components could give a project a more speedy delivery than if all the code is to be written from scratch. This is often also the case with development using COTS, but this adds the extra cost of obtaining and using it in your project.

3 Selection of OTS components

Researchers have proposed several structural, formal or semiformal models to choose OTS components. Some of the OSS evaluation methods will be presented in Section 3.3. Regarding the COTS selection process the authors (Mohamed, Ruhe, & Eberlein, 2007) present what they chose to call the "General COTS Selection (GCS)" process which comprises of five steps:

1. Define evaluation criteria for the COTS
2. Search for COTS components
3. Filter the results based on some "must-have" requirements
4. Evaluate this filtered result-list
5. Do an analysis of the output of step four and select the COTS that fit best with the evaluation criteria.

Even though it is not any commonly accepted COTS selection method they conclude that most formal methods share these five steps.

Empirical data seems to contradict what is proposed in the formal methods. The normative methods presented in literature seem to be seldom used. Consider the following quote from (Li, Conradi, Bunse, Marco, Slyngstad, & Morisio, 2008):

"Component selection: Integrators select OTS components informally. They rarely use formal selection procedures"

This is one of the "10 facts" presented as a result of an international survey of 133 projects from 127 companies. The findings suggest that the integrators select components in an ad-hoc manner, using personal experience or web-based search engines.

Some firms exist that specializes on helping other companies, select, evaluate and integrate OSS software. Navica is such a company (Navica Inc., 2008). They deliver service in form of strategy, implementation and training services. Selection(requirement analysis, identification of candidate OSS and evaluation) is part of the strategy service, lecturing in OSS best practices is part of the training service and helping the organization to roll-out the OSS is part of the implementation service. They have served many companies during the last year including SugarCRM, Emulex, Red Hat, and the US Department of Defense.

The selection of an OTS component is not an easy task but it could be revised in three phases, namely identification, evaluation and choice. These will be discussed in the forthcoming sections together with a small introduction to formal evaluation models.

3.1 Identification

Companies searching for OSS components often look for either reusable code or reference examples (Umarji, Elliott, & Lopes, 2008). Reusable code is code they could use without any modification such as an implementation of a search algorithm, a

wrapper, a parser, GUI widgets etc. A reference example, in contrast, needs modification to fit the component's purpose, or the reference example could just be used as a piece of code that shows how a problem could be solved. Examples of this include how to implement a data structure or algorithm, how to use a library or simply looking at similar systems to get ideas on how to implement their own system.

How do companies search for OSS components? An online survey performed in 2008 (Umarji, Elliott, & Lopes, 2008) showed that only a small fraction of the respondents (11/69) used code specific search engines when searching for source code. Most of the respondents (60/69) used general purpose search engines like Yahoo! and Google. About half of the respondents (34/69) used project hosting sites like SourceForge. This is further documented in an empirical study performed in Chinese software industry: *"…our results show that developers used Google more frequently than OSS project portals"* (Chen, Li, Ma, Conradi, Ji, & Liu, 2008).

Also, it seems that personal experience, others' recommendations and social networking may play a vital role in the identification process (Merilinna & Matinlassi, 2006). Personal experience covers both the case when the adaptor has used the component before and when he has read about it in e.g. a book, an article or at web-sites. The mentioned study of the Chinese software industry further states: *"…our results reveal that experience sharing between persons in different organizations was not common."*(Chen, Li, Ma, Conradi, Ji, & Liu, 2008). If this is the case in other countries too is difficult to draw a conclusion about. Therefore research on the subject is necessary.

Source for components	**Examples**
General web-based search engines	Google, Yahoo
Specialized web-based search engines	Google code, Merobase, Koders
Project hosting sites	SourceForge, CodeHaus, Tigris
Language specific project hosting sites	Java.net , Java-source, Open Symphony, CPAN, PHP Classes
Personal experience	Read about it, used the component before
Others' recommendation / social networking	Friends, colleagues

Table 1: Identification methods

Table 1 shows a summary of several different sources of identification. It is not a complete list of all sources for components but it lists some of the possible ones together with some examples.

3.2 Evaluation

When evaluating COTS products, there are three main strategies according to (Mohamed, Ruhe, & Eberlein, 2007):

1. Progressive filtering – Starts out with a large number of COTS components reducing these through successive iterations of product evaluation cycles until only a few decent components are left.
2. Puzzle assembly – This strategy assumes that the COTS-based system needs several COTS & other products to fit together like pieces in a puzzle. This further means that component that could fit in isolation does not necessarily work together with others. The strategy therefore proposes considering the requirements of each component while simultaneously considering the other products in the "puzzle".
3. Keystone identification – The third strategy starts by identifying a key requirement and then search for products that satisfy this requirement. In this way time is saved by quickly eliminating products that do not satisfy the "keystone".

Some similarities are present when evaluating OSS components. It might be wise to define some evaluation criteria (Cruz, Wieland, & Ziegler, 2006). Table 2 presents some of the criteria the developer team could consider when evaluating a component.

Evaluation Criteria	**Examples**
Functional	Features of the software
Technical	Number of bugs, number of feature requests, frequency of changes
Organizational	Number of developers, testers and users
Legal	Licensing issues
Economical	Migration costs

Table 2: Evaluation Criteria

The functional criterion is of course important, since it is why the project team is looking for the component in the first place. The technical and organizational criteria prove important to get an understanding of the "liveliness" of the component, and how thoroughly tested the component is. Legal issues are important if the software is to be released as a commercial product, this requires a license which allows such release of software. And finally the economical criteria are also important so that an analysis of the cost to create the component from scratch is set up against the integration cost if an OSS component is selected.

However there may be a mismatch between the listed/documented functionality and what actually is implemented and fully working. Therefore the adaptor often has to do some testing and prototyping with the component in order to both validate the documented functionality and to verify the possibility to adapt the component to fit the project/system at hand. While prototyping the adaptor also get an insight on how easy the component is in use and consequently how good the usability of the OSS component

is. In addition when studying the source code a first impression could be made of the quality of code documentation and modularization of the component, which in turn has an impact on the extendibility of the component.

3.3 Formal OSS evaluation methods

Several OSS evaluation models have been proposed in literature like the Open Source Maturity Model (OSMM) (Golden, Making Open Source Ready for the Enterprise: The Open Source Maturity Model, 2005), the Open Business Readiness Rating (OpenBRR) (BRR 2005 - RFC 1, 2005), the Qualification and Selection of Open Source Software (QSOS)(Atos Origin, 2006) and Open BQR(Taibi, Lavazza, & Morasca, 2007). Open BQR builds on experience of the other mentioned models and further address internal and external product qualities in a, according to the authors, more adequate way. The method further tries to minimize time wasted in evaluating components that quite obviously will be pruned due to low or non weight. Open BQR is divided into three phases:

1. Quick Assessment Filter
2. Data Collection & Processing
3. Data Translation

During phase one of Open BQR evaluation scope, target use, external qualities (number of bugs, bug fix rate etc.), internal qualities (code metrics), product support and a weight on basis of the existence of required functionalities is addressed. Phase two prunes the components that are given a weight below a user-specified threshold. Next the remaining components are measured, normalized to a scale between 0 and 100 and finally assessed as a weighted sum of the results of each of the single areas. This gives the required information for phase three that consists of a visualization of the result. See Figure 1 for an example of such a visual representation.

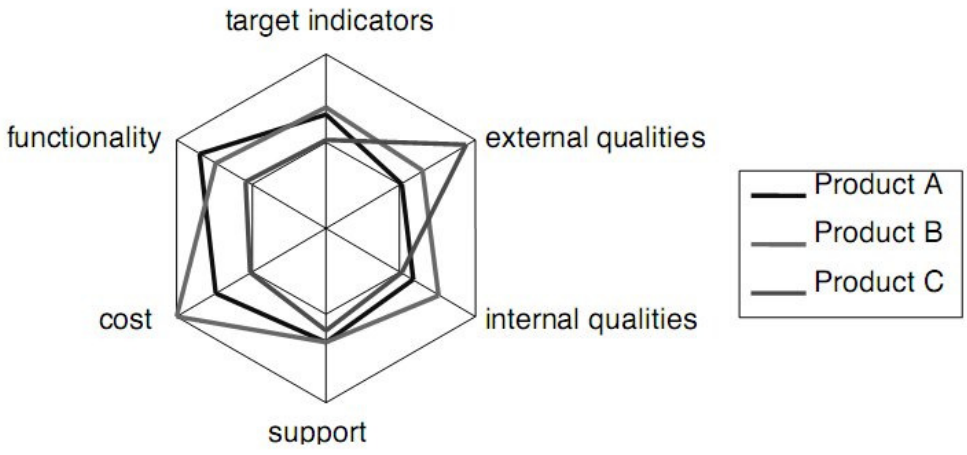

Figure 1: Open BQR evaluation: visualization of the comparison of three products from (Taibi, Lavazza, & Morasca, 2007)

Another model that is developed is the SQO-OSS Quality model (Samoladas, Gousios, Spinellis, & Stamelos, 2008). This model main difference from the others is that it does

not evaluate functionality, but rather focuses on maintainability, reliability and security. It was constructed with focus on automation and minimization of user interference. The model uses four categories for aggregation: Excellent, Good, Fair and Poor. In addition a definition of three profiles (one for each of the three first categories) is produced so that a component could be placed in one of those categories or in the last (Poor) if none of them fit. As an example the high level quality maintainability is judged by the lower level attributes (or sub-attributes): analyzability, changeability, stability and testability. Each of the sub-attributes is evaluated to the three profiles before the final conclusion on the high level quality is produced. This implicates that if maintainability is to be characterized as Excellent, all of the sub-attributes needs to be characterized as Excellent. In the same fashion to characterize the whole product quality as Excellent, maintainability, reliability and security has to be characterized as Excellent. Based on this the product quality of different component a more educated choice could be made when choosing the component to use.

As for OSS also several models for COTS selection has been proposed during the last two decades. Mohamed, Ruhe and Eberlein have listed and compared 18 of these, but they conclude that it is not an easy task to choose the best fitted selection method (Mohamed, Ruhe, & Eberlein, 2007).

3.4 Choice

From the identification and evaluation some conclusions should be possible to make about which component(s) that are best appropriate to choose. It may not always be possible to find a component that fulfills all of the evaluation criteria defined. Therefore instead of finding the perfect fit the development team finds the component that seems to cover most of the functionality and most of the other important criteria. This is what is referred to as a “best-fit” choice. This strategy could prove time consuming, since many components have to be considered. But its advantage is that it finds the best possible fit. On the other side, another approach is the “first-fit” strategy, this means choosing the first component that is suited to do the job, and neglect identifying and evaluating other components. This has the obvious disadvantage that better components may be dismissed, but has an equally considerable advantage in time saved by not looking at several candidate components.

4 Challenges in OSS development

A major challenge when using OSS/FS is how to integrate the software/component with existing architecture and components and how to maintain the components as part of an existing application (Merilinna & Matinlassi, 2006). In contradiction to development using COTS, you do not have any vendor that is responsible for the quality of the software, this responsibility falls on the user (Golden, Succeeding with Open Source, 2005).

Which component is the best fit for the project's purpose is also a large challenge. One could ask for example how important is it that the component fulfills all the functional requirements in contrast to only some of the security requirements? Or another question: how should two components that are good in different dimensions be evaluated against each other? These questions are not easily answered.

Another challenge is to get an understanding of the OSS component's license and all other issues that may arise when using several OSS components. For example, two or more OSS components could be incompatible with each other. Further in many cases it may not be desirable to release the source code with the software to be developed. In addition, as mentioned in Section 2.3, the enormous amount of different licenses could force the developer to use some time studying what the specific license implies for the project.

Depending on the liveliness of the community of the OSS components, new updates and fixes will be released from time to time. When and how often the integrator should upgrade his project is an issue to consider. It may not be as easy as just "installing" the new update, the new update needs to be integrated with the project and may cause a significant overhead cost. The project team therefore has to carefully consider when and how often these new releases should be integrated. Actually many companies strive to treat OSS components as closed source products (Merilinna & Matinlassi, 2006). This is desirable so that the company could reduce maintenance costs. Therefore companies tend to freeze the current OSS component version to avoid any ripple effects in case of frequent updated releases. This has naturally also its disadvantages, for example in the case of a new and better patched version or a new release with desirable functionality.

Choosing components on the basis of previously use is not necessarily always the best option. This could be seen upon as a "first-fit" solution, and that may not necessary be the so called "best-fit", there may be other components out there that does the job at hand on a better or more efficient way. However it is clear that time is saved if the adaptor could choose a component based on successful previous use. Also, by mainly choosing on earlier experience one could miss out on components that either was not found the first time around or have started to exist in the void between the previous use and now. These components could be of better quality or with more desirable functionality than the one that was chosen earlier.

As the amount of OSS components grow, the chance of finding components that is satisfactory for the adaptor's use increases. This may lead to that more and more OSS components are integrated into a solution with in turn could increase the maintenance costs. A hypothetic issue is therefore that a company has too many OSS products in different solutions and that it consequently becomes hard to be able to effectively maintain the systems.

Another hypothetical question is how the competence and knowledge of the employees is affected by CBSD. A decreasing amount of the software is developed from scratch instead they use finished components for large fractions of the functionality of the system. Do they learn enough of the new technologies to evolve to be better programmers? Or do they just become searchers for components and do not get to practice coding in a sufficient manner? These questions are not easily answered.

Many formal methods for OSS selection exists, as presented in Section 3.3, but it seems very few of them actually are in use. It seems that companies do OSS selection in an "ad-hoc" manner instead of using the formal OSS selection methods. One could argue that this may be because the methods presented in literature are:

- Not known to the companies
- Too extensive and time consuming
- Not thoroughly tested in the industry

What does developers and companies do in practice when they plan to use OSS components? How do they do this selection process? Even though many formal processes and guideline has been proposed in literature and through research there does not seem to be a common model or method. Therefore more empirical research on the subject is needed.

Part 3 Research

This part consists of two chapters. The first chapter will give the motivation for the research and state the research questions. Next, Chapter 6 will present the research method including information regarding sampling and analysis of data.

5 Research design and context

The Norwegian software industry contains about 12 000 active companies (StatBank Norway, 2008). Most of these however are small so only 10% of the companies have more than 5 employees. In contrast to big international companies with hierarchic structure, most of the Norwegian companies have a flat structure. Because of the short distance from developer to management (short decision-chains) it may be easier to perform a bottom-up approach on OSS integration. Table 3 lists some key figures for the Norwegian ICT[1]-sector.

Sector:	# Employees	Total Turnover in M NOK	Total Value Creation in M NOK
ICT-industry	10 856	29 696	9 373
ICT-commodity trade	11 843	46 641	7 063
ICT-telecommunication	12 775	60 331	23 143
ICT-consultancy	38 096	54 839	28 519
ICT-sector Total	**73 570**	**191 507**	**68 097**

Table 3: Key figures Norwegian ICT-sector

5.1 Motivation

There has not been much research on how selection of OSS components is done today (Umarji, Elliott, & Lopes, 2008). In addition to the evaluation criteria shown in Table 2: Evaluation Criteria, (others) personal experience and network resources seems to be important aspects when choosing which OSS component to use (Merilinna & Matinlassi, 2006). Formal methods presented in research literature do not seem to be too widely used, so what is done in practice? This is what will be studied in this research.

5.2 Research question(s)

The following research questions have been formed:

RQ1: How do Norwegian software companies identify possible OSS components?

RQ2: How are the identified OSS components evaluated?

RQ3: How is the final choice of the OSS components accomplished?

The three research questions together form what could be named the OSS selection process consisting of identification (search), evaluation and choice. In research some ambiguity in the use of the term “select” is present as some use the term it to cover the whole process while others mean only the final choice.

[1] ICT - Information and Communication Technology

Finally the fourth research question is:

RQ4: Which resources do Norwegian software companies use or to aid the OSS selection process?

In addition to studying the selection strategies a side-effect of the study will be some input on to what degree open source products in Norwegian software industry is used today and which OSS components that are popular.

6 Research method

To answer the research questions several semi-structured interviews was performed in the Norwegian software industry. In total 16 companies was contacted and asked if they have a developer that could spare an hour for an interview concerning an assignment on OSS selection strategies at NTNU. Five companies agreed to participate in the research. Interviews are according to Oates effective tools to generate data with detailed information (Oates, 2006). It allows open-ended questions and does not just provide pre-defined, or closed, answers like a questionnaire do. In addition more explorative questions could be used and as the interview goes along follow-up questions could be asked to gain more qualitative information, and clarification questions could be asked to make sure one understood the answer fully.

Myers and Newman summarize several potential pitfalls, difficulties and problems with conducting qualitative interviews in IS research (Myers & Newman, 2006). One issue is the artificiality of the interview. Since the interviewer often is a complete stranger to the interviewee and together with time constraints this result in incomplete answers. Another issue is the Elite Bias, a term introduced by Miles and Hauberman (Miles & Hauberman, 1994). The elite bias comes to being by only interviewing the most important people of an organization or company, their "stars". This may cause an unbalanced picture since none or few of the low-status informants are considered. A third problem is the ambiguity of our language, and in some cases it may be difficult to be sure that one have a perfect understanding of what the interviewee is trying to tell.

To address these issues some guidelines should be considered (Myers & Newman, 2006). Making a good first impression and carefully explaining the purpose of the interview ahead of the questioning could reduce the artificiality of the interview. Further, maintaining the confidentiality of the conversation/transcripts is important. Planning interview with a semi-random group of interviewees is a way to reduce the elite-bias. Finally by being flexible in the questions and use mirroring[1] in the questions and answers could prove to reduce the misunderstandings of the language ambiguities.

The interviews was taped and transcribed to more easily cover all details that may be of importance to the research. By using a voice recorder one avoids having to make extensive notes. However some notes were made so that good follow up question and probes could be formed. Without notes at all such immediate questions may easily be forgotten, especially in the cases were the interviewee has a long and cumbersome answer.

Observation from within a company and project may also have provided great research material, but due to the difficulties of gaining access and support for such an observation this research method was not considered as a feasible method.

[1] "Mirroring is taking the words and phrases the subjects use in construction a subsequent question or comment: mirroring their comments" (Myers & Newman, 2006).

Using Questionnaires is also a research method to gather data. They could be used to collect data from a large number of people and are especially good for closed questions, questions that easily could be answered on a scale from x to y (Oates, 2006). For example a statement could be made and the respondent answers on a scale from "strongly disagree" to "strongly agree". Their main disadvantage is that the researcher cannot check the truthfulness of data and it may be difficult to pick up misunderstandings (in either question or answer). Questionnaires was not used in this research since it was desirable to go in depth in issues and get more qualitative data, instead of quantities data that results from a questionnaire.

An interview guide has been developed by Øyvind Hauge and Claudia Ayala and has been reviewed in several rounds. When I first joined the project the interview guide was more extensive and covered more details about maintenance and evolution. It was decided that the guide was too comprehensive and therefore these details were deleted and an increased focus on the actual selection process was implemented. Claudia Ayala conducted three pilot interviews in Spain and it seemed that the interview guide was satisfactorily with minor adjustments and the final version is attached in Appendix B. This version was also translated to Norwegian to ease the interview process of Norwegian respondents. The interview guide has questions related to OTS components; that is both COTS and OSS products. However, only the OSS parts will be considered in the discussion in Part 4. Table 4 shows the relationship between the questions in the interview guide and the research questions of this study.

Question number in interview guide	Main theme covered by the question	Research question
3,4,5,6	Identification	RQ1
7,9, 10, 11, 12,13	Evaluation	RQ2
8, 14, 14*	Evaluation & Choice	RQ2 + RQ3
16	Internet resources – Identification and evaluation	RQ1 + RQ2
17,18,19	Internet resources	RQ1 + RQ2 + RQ3
20,21,22,23,24,25,26,27	Personal & Social resources	RQ1 + RQ2 + RQ3

Table 4: Mapping between questions in interview guide and research questions

6.1 Sampling

The term Sampling is closely related to the term population (Robson, 2002). While a population refers to all cases, for example all inhabitants in Norway, a sample refers to a selection from this population. In this study the Norwegian software industry could be seen upon as the population. The sample drawn from this population is developers working in Trondheim and a total of five companies were picked. This could be seen upon as a case of convenience sampling, picking developers nearby as respondents to avoid much travelling or having to perform the interview over telephone.

The research started with calling the companies by their general phone number. Most of the companies were found due to my own knowledge of the Norwegian software industry, and some additional were found by a general search for IT companies in the

yellow pages[1]. Most of those who were contacted were positive to the inquiry of a possible interview. Others wanted a written request by e-mail. Some did not have time to attend and a few said they did not use that much OSS, and therefore was not that interesting for the study.

Table 5 shows the companies that participated in this research and Table 6 gives some information on the respondent from each interview.

Company	Type of company	Company ownership	Business area	Total #Employees
Alpha	Stand-alone	Public	Language Technology	~10
Beta	Subsidiary	Private	Consultant	200+
Gamma	Stand-alone	Private	Consultant	~20
Delta	Stand-alone	Private	Consultant	63
Epsilon	Stand alone	Private	Consultant	21

Table 5: Company information

Company	Respondent work experience	Respondent Position	Respondent Educational Degree	# OSS projects participated
Alpha	7 years	Manager of development	Cand.mag. Computer Science	2
Beta	4 years	IT Manager	Master Technology	~12
Gamma	11 years	Senior Consultant (Developer)	Master Technology	10
Delta	9 years	Chief scientist	Master Technology	~20
Epsilon	9 years	Technical manager	Bachelor Software development	7

Table 6: Respondent information

As seen from Table 5 and Table 6 most of the interviewees were from private consulting firms, had a master degree and had worked on several OSS projects. The companies they represented were ranging from 10 to 200 employees. The respondents had an average of 8 years of experience working in the software industry.

6.2 Analysis of data

The analysis of data started with the transcription of the five interviews. This was a tidy and time consuming process, with an estimate of 4-5 hours of writing for each hour of tape. After this the transcripts was printed they were analyzed in terms of which of the responses in the transcripts that answer each of the research questions. This was aided by Table 4 and by using different color of markers for each of the RQs.

[1] http://www.gulesider.no

After this followed much “copy paste” and structuring of the responses in a Word document to ease the process of finding similarities and differences between the answers given by the interviewees. It was at times difficult to get a full overview of the about 50 pages of transcripts, but given enough time some results and similarities were identified. Next these results and similarities had to be generalized into concepts that could try to explain the phenomena or procedures that were elaborated during the interviews and these results will be presented in the forthcoming part of this report.

Part 4 Results, Discussion and Conclusions

The structure of this part is as follows. Chapter 7 covers the results from this study and Chapter 8 gives a discussion of these results. Further, Chapter 9 considers some limitations and an assessment of the validity of this study. Finally, Chapter 10 gives a conclusion of the report together with suggestions for future work.

7 Results

This chapter will present the results from the interviews. The chapter has five subsections. One for each of the research questions, 6.1 through 6.4, and a final subsection with other findings in 6.5. This last subsection presents findings that are not completely related to the research questions, but which still is interesting findings and could lead way for further OSS research.

7.1 RQ1: Identification

The first research question was concerning how Norwegian software companies identify OSS components. My insight is that this identification process is twofold. One side is proactive in maintaining a general awareness of what is out there in the OSS communities and carefully observe market trends, see the quote from the left hand side of Table 7. The other side is reactive by finding solution to challenges or problems as they occur. In this reactive reaction the first proactive part could prove very useful in finding components that may be suitable to help solve the issue at hand, see the quote from the right hand side of Table 7.

Proactive:	**Reactive:**
Epsilon: *"For example we have been looking at 'Ruby on Rails', but it has some shortages in functionality compared to what is default in Hibernate […] Then we wait. It is on our radar, but we wait"*	Gamma: "We use Google much, if one have a specific problem to solve, then Google is the thing to use"

Table 7: The twofold identification process

Considering the proactive part of the identification process the companies maintain the general awareness through various resources. One of these is to either read blogs and Internet sites such as TheServerSide[1] or subscribe to an RSS feed from these kind of sites. TheServerSide is an Internet news site, for enterprise Java architects and developers, which daily publish articles of various kinds and often writes about state of the art OSS products and market trends. Another resource to aid the proactive part of the identification process is others experience with OSS software components, together with own experience from earlier projects. A third resource to maintain a general awareness is to attend conferences, for example JavaZone that is hosted annually in Norway. This is an excellent arena to pick up what is moving in the OSS arena by

[1] http://www.theserverside.com/

listening to seminars and talking to colleagues. Furthermore printed literature in form of e.g. O`Reilly books also supports the proactive part of the identification process.

The second part of the identification process is the reactive part. This is a phase where the companies for example experience a problem in their project which they do not immediately know how to solve. Then a quick Google search could prove useful to see if others have experienced the same problem, or if there actually exists an OSS component that solves the problem. Another example may be when a company enters a domain where they do not have that much experience and therefore tries to identify a component that may ease their efforts on understanding the problem domain.

Several types of resources are used to aid this twofold process and they could be divided in two main categories:

1. Internet resources.
2. Personal and social resources.

These will be more thoroughly discussed in Section 7.4.

7.2 RQ2: Evaluation

Moving on to the second research question considering the evaluation of the identified OSS components; this evaluation does not seem to be very rigid in terms of strict evaluation criteria and careful analysis. Indeed some evaluation criteria may exist but the evaluation is done in a more prototyping fashion, downloading the component and testing it “in action”. In addition to this an assessment of the present documentation in form of e.g. FAQ, How-to’s, quick start or getting started is often performed. Further this documentation aids the process of assessing the maturity of the component, which also could include a quick analysis of the code metrics, number of commits and the change log. Additionally, previous use of the component could be a vital part of the evaluation process if they have had positive earlier experience with the component. The evaluation could be discussed in two dimensions, namely what the companies do to evaluate components, and which criteria that seems to be important for the evaluation.

Prototyping is the clearest example of what companies do to evaluate components. All of the respondent companies mentioned this as an important part of their evaluation. It is an easy way to check if the component does what it is supposed to do, and further do a quick analysis the usability, performance and other criteria that could be important for the development of the project. Several of the interviewees stated that the evaluation process often differs in how to evaluate small vs. larger components. Smaller, less important components could easily be downloaded and tested in such a prototyping fashion as described. Larger, more important components, often face a more rigorous and extensive evaluation. The same seem to account for minor libraries vs. fundamental frameworks. In the latter a more serious assessment of e.g. user mass and OSS maturity is performed to evaluate the components. One of the respondents reported that he has participated in the development of a tool to support some of the activities in the

evaluation. He describes the tool as a search engine which use change logs from the version control system (e.g. CVS[1], SVN[2]) to produce relevant statistics. These include the number of developers contributing, how this number develops over time and if there is part of the code that only a few developers work on(core developers), or pieces that many persons has worked on. The respondent said:

"But this is a resource I use to find out who that has been contributing to the project. Is it a "one-man-show" or… And it may help to analyze what that actually has been done.

Evaluation Criteria	Alpha	Beta	Gamma	Delta	Epsilon
Documentation	X	X	X	X	X
Component Maturity		X	X	X	X
Number of component followers		X	X	X	X
Change log/Number of commits (e.g. pr week)		X	X	X	X
Ease to use	X		X	X	
First impression of website	X	X		X	
Past Experience		X	X	X	
Forum support (Especially: response time)		X			X
License			X	X	
Support from a large Company (e.g. Redhat, SpringSource)	X				X
Performance	X			X	
Gut feeling				X	X
Consult literature				X	X
Reference documentation – Others refer to the component				X	

Table 8: Evaluation criteria used by the companies

Considering evaluation criteria that seem to be important, a wide range of these exists as could be seen from Table 8. But these are not necessarily criteria which are used in every evaluation of OSS components. Several of the respondents noted that these could change from project to project and from OSS component to OSS component. However, a criterion that seems to be important to the evaluation in almost any case is the quality (or availability) of documentation. Different kinds of guides on how to use the component are evaluated to see if the component serves the purpose it is supposed to solve. This documentation is important as an introduction to the component and what it could be used for. Next an important criterion is the component maturity, since an adoption of an immature component could, in a worst case scenario, result in adopting a component that is not maintained and further developed. Associated criteria to this are assessing the number of component followers and the number of commits. Epsilon summarizes most of the criteria given in Table 8 in the following statement:

[1] http://www.nongnu.org/cvs/
[2] http://subversion.tigris.org/

”It usually goes like this. You download the component and test it, you start asking questions in forums. You check the response time on the response. Does it [the component] have a large contributor present? And you may look a bit at the change log to see when code was committed to the project, and from there get an impression if it is maintained properly.”

A comment on one of the other criteria listed in Table 8 should be addressed. The fact that only two of the respondents mentioned that the license of the component had influence on the evaluation may be a bit misleading. If a question about how important the license was for the evaluation and choice were included in the interview, I suppose the majority of the interviewees would have answered that it was important. Especially since the amount of GPL components out there is large and companies may tend to refrain from these since they want to avoid the “copyleft” effect.

Additionally it is interesting to comment on the lack of a couple of criteria in the table. Maybe most importantly “functionality” and “technology” is seen lacking from the overview. Functionality of the component is just the reason why OSS components are evaluated in the first place; it serves some functionality that is wanted by the companies in a specific project. Therefore it could be seen as the most important criterion altogether even though this was not necessarily explicit mentioned in the interviews. Connected to the functionality it is important to also consider the technology platform. It serves no use to look at Java specific components in a .Net project. These two criteria may not be explicit mentioned by the respondents due to that they may think they are too obvious to mention.

7.3 RQ3: Choice

Advancing to the third research question; the final choice of evaluated components is analyzed. Here the respondents had differing strategies. Some of the respondents told that after the evaluation the choice was just to choose the one that seemed to be the best fit, in that case the evaluation and choice could be seen as a single phase. In case of having only one component under investigation a more first-fit choice will be made. Others told that in addition to the evaluation a final go has to be confirmed by someone higher in the organization, which implies a more top-down approach.

Eying the first-fit / best-fit type of choice, the obvious is that successful previous use implies that the component will be used for similar tasks again, and the identification and evaluation phases are dropped, which reduces the total selection effort. This is a form of first-fit choice, since it does not evaluate which other candidate components that could do the job with equally or even with better results.

The other part of choice strategy is the more top-down approach, where someone higher in the company hierarchy has to give the final go. My impression is that this is more applicable to the final choice of a COTS component than an OSS component, since the costs associated of the commercial variant often remarkably exceeds the costs associated with the OSS variant.

7.4 RQ4: Resources

The last research question is what resources which are used to aid the selection process. My understanding is that these resources are split in two main categories: Internet resources and personal and social resources. These will be addressed in turn next.

7.4.1 Internet Resources

The different Internet resources that are used by the companies during this selection process are listed in Table 9. As from the table, Google is an important tool in identifying components. This was also noted in Section 7.1 as part of the reactive identification process; having an immediate need for a solution in a specific context. An "easy" problem could result in a Google search for a solution, while a more complex issue maybe has a better chance of resolution by a question in the components forum.

Type of Internet resource	Alpha	Beta	Gamma	Delta	Epsilon
Google	X	X	X	X	X
Javabin[1] – Norwegian Java users association		X	X	X	X
News sites (e.g. TheServerSide, ,D-zone[2])		X	X		X
Forums		X	X		X
Blogs		X		X	
Project pages(e.g. www.springframework.org)			X		X
SourceForge	X				
Usenet	X				

Table 9: Internet resources used in the identification of OSS components

It should be noted that it was not asked directly in the interview if they used these different resources, the table is just a summary of which of the resources that were mentioned by the different companies. Probably most of the companies use SourceForge in some way during the selection of components even though only one of the respondents mentioned it during the interviews. This is highly probable due to the fact that many of the OSS components do not have own web sites, and use SourceForge for webhosting of the project.

Three of the companies told that they actively visited TheServerSide. Since the start in year 2000 this news site has grown to be the largest independent Java community in the world. Epsilon characterizes the site as:

"… and TheServerSide is more like the entry gate that tease your curiosity."

TheServerSide is an example of the proactive part of the identification process as discussed in Section 7.1. It is a helpful site to observe what moves in the OSS

[1] www.java.no – Host the annual JavaZone in Oslo, Norway.
[2] http://www.dzone.com

communities, and getting a general view of which components that could prove helpful in a future project.

As a concluding remark of Internet resources it can be observed that most of the respondents considered the usefulness of Internet resources as crucial and even essential for any development with OSS components. Consider the quotes given in Table 10. It is clear that the Internet resources are important and could be seen upon as a requirement for doing CBSD.

Respondent	Response to the usefulness of Internet resources
Gamma	*"Yes, it is extremely useful. One is in many ways dependent on it. If not the task of staying up to date on everything is way too big, so you could say that it is a question of existing or not existing that you have it available. It is extremely important, especially when information is put out on the Internet, that it is done in a proper way so that it shows that it is well documented. This helps build trust [in the component]"*
Epsilon	*"Frankly, it is everything. It is that way we get information and get interested in things, and it is that way we solve problems and communicate. So it should be clear...If it had not been for that [the Internet] then we had not used that many OSS components, then we would probably have bought the components instead. "*

Table 10: Quotes considering the usefulness of Internet resources

7.4.2 Personal and Social Resources

The Internet resources are not the only resource used by the companies. Several of the respondents promptly stated that personal and social resources may be of equal or even more importance than the Internet resources. As some of the respondent mentioned you cannot trust everything you find on the Internet. There are many "silly" people on the Internet, and it may prove easier to trust a person from your social network than just a person with a nickname and an avatar on a random web site. Important personal and social resources include personal experience (previous use) and user groups like Javabin that host conferences like JavaZone.

A successful usage of a component in an earlier project considerably increases the possibility to use it again in a later project. Others recommendation is also considered and many of the respondents said that they tried to stay updated with the market trends which further supports my insight about the proactive part of the identification process.

In the cross-section between social and Internet resources are web sites like is OHLOH[1]. The site uses the version control, mailing lists and bug database of the different OSS projects to offers code metrics, reviews, lists contributors etc. It had over 21000 projects in the database by the end of November 2008.

[1] http://www.ohloh.net/

7.5 Other findings

In addition to answering the research questions a better view of which OSS components that are in use by the different companies and from there a small pointer of which OSS components that are popular today is a result of this study. Secondly some interviewees commented on the lack of documentation in OSS projects. Thirdly the formal methods presented in literature are seldom used, as could be seen from the results from the research questions discussed earlier. Finally some of the respondents wished there was a web site that could aid the selection process in gathering several OSS projects in categories together with others experience with using the component.

Company	OSS technologies used in the project
Alpha	• OpenOffice SDK • WIX • GCC • nAnt • nUnit • Nullsoft Scriptable Install System
Beta	• Spring • div commons libraries
Gamma	• Spring • div commons libraries(e.g. commons.lang and commons.log) • AOP Alliance • Spring Web Services
Delta	• Spring • Hibernate • Lucene • log4j • MySQL • Google Web toolkit • JQuery
Epsilon	• Spring • Hibernate • Postgres • Java Server Faces • Facelets • RichFaces • JasperReports • Myfaces Tomahawk

Table 11: OSS technologies used by the different companies

Table 11 lists the OSS technologies in use at the projects discussed during the interview. It shows that four of the respondents reported usage the Spring Framework in the project discussed. The Spring Framework has during the last years become a well known application framework for Java development. It includes a common abstraction layer for transaction management, a JDBC abstraction layer, AOP functionality and much more features to ease the development of the application (SpringSource, 2008). In addition it could be seen from the table that Hibernate is used by a couple of companies. This is, as

the Spring Framework, an OSS component with increasing popularity. It is an object/relational persistence and query service that claim to be powerful and cause high performance (Red Hat Middleware, 2006). For more information, see Appendix A for a full overview of the web pages of the different components.

A few of the interviewees confirmed the often commented fact that many of the OSS projects out there are not sufficient documented. Even though this was not part of the research question its worthwhile mentioning, and Table 12 gives a few quotes concerning this issue.

Respondent	Quote
Alpha	*"…you could say that on a general level OSS presents itself on a rather indulgent way. Roughly speaking it is often one or two project founders that use all their spare time to hunt down bugs in their project, and setting up decent web resources is not prioritized. So the general tendency is that it [OSS projects] actually has poor documentation."*
Gamma	*"… there are some [projects] that are not that good documented on the 'OSS-side'. What would make things much easier was if more of those that offer OSS-libraries had worked more with the documentation because it is often not satisfactory."*
Beta	*"… And the documentation of course. Because it is frequently there [the documentation] that there is a lack of in the OSS-scene [in comparison to commercial products]."*

Table 12: Quotes concerning lack of documentation in OSS projects

On the other side the respondents noted that this is not necessary the case, some projects out there has good documentation and resources at their website. An example that were presented in this context is WiX (Microsoft Corporation, 2008), which is a toolset that builds Windows installer packages from XML source code. This website provides the reader with a manual, a tutorial, a FAQ and a very informative mailing list.

After analyzing the interviews it became quite clear that none of the participating developers used any of the many formal methods presented in research literature (see Section 3.3). It would have been interesting to ask the developers if they even knew that such formal OSS evaluation methods existed, but this was not done in this interview. The selection process is more of an ad-hoc fashion and the word of mouth, both in real life and on the Internet, seems to have a great influence in the companies' selection of components. This was promptly stated by Delta:

"We do not have a formalized process for it [the selection of components] really. It is rather ad-hoc although we use the Internet and we use… we look at what other do, recommends and has experience with."

Considering what is missing a few of the respondents said that an online service that kept track of open source components and companies experiences with using it could be

a useful addition to the selection process. A premise for this is that it is updated and that the users insert their experiences into this database, Delta supports this by saying that the experience with such “knowledge databases” is that they do not work because people do not use it, and the knowledge contained becomes old and outdated. Epsilon states that in such a site he would have liked to get some statistics about response time in the forums, how the code is maintained and total lines of code for each project. That would ease the selection process by having a better general view of what components that is available for the task at hand. As mentioned Ohloh is such a site, but a more extensive, updated and well known site is wanted by several of the companies.

8 Discussion

In this chapter the results presented in the previous chapter will be discussed. The main findings are that the identification process is twofold, the evaluation is of a prototyping fashion and the final choice is made in a "first-fit"-like fashion. Resources that aid this selection process are of two main types: Internet and Social/Personal resources. Further formal methods do not seem to be in use and developers lack a central place to gather experiences to aid the selection process. These issues and a discussion around the quality of OSS documentation will be discussed in the following sections.

8.1 The selection process

In today's technology age, staying updated on new technologies and market trends is important and maybe even crucial for success, and the intellectual capital of the employees has to be developed. Thus that the identification process is twofold in terms of a proactive and reactive part is not that strange. The proactive part is part of the developer's constant hunt for knowledge and ways to do things more efficiently and therefore save time and money. With the enormous amount of information and content on the Internet today, searching on Google and other search engines for help in case of problems is a natural way to ease the development effort, hence the reactive part of the identification process.

The finding that the evaluation is done in a prototyping manner instead of using extensive and rigid evaluation methods and criteria could be explained by time constraints in the projects. Therefore, using an extensive amount of time on evaluation is not desirable. Additionally many developers out there are used to a "learning by doing" approach to development, and what better way is there to do this than simple trying the component. Downloading the component and creating a simple prototype could be done easily and may reveal strength and weaknesses of the component relatively quick in contrast to that of rigid evaluation methods with many evaluation criteria. In the end, it is the functionality of the component that is important, and the best way to test this is simply by creating a prototype.

Also the choice strategy may be explained by companies wish to save time and money. A best fit strategy is much more time consuming since a wide range of components has to be evaluated before the most fitting component is found. It is also quite obvious that if a company has had good experiences with an OSS component in an earlier project, the component will be used again for similar tasks, and therefore the identification and evaluation is skipped and time is saved. But this kind of first-fit approach may cause the developers to overlook that there may exist substantial better components available, either in form of a component that has better performance, security or similar attributes, or components with additional and desirable functionality. Additionally, a thing that may have influence on the choice is adaption by a large company, e.g. Microsoft or Red Hat. This is something that helps build confidence in a component. When Alpha was looking for an installer package and noticed that one of their candidate components was used by Microsoft in the Microsoft Office program suite that had a big

impact on their final choice. It is not necessary that such a large company adopts a project, but it clearly shows that the component is mature and that it solves an issue in a satisfactory way.

Considering the resources the companies use to aid the selection process, the findings of this research is that personal and social resources are equally important to Internet resources. Using the resources available in the company and the social network is very helpful to the whole selection process. Earlier experience with a component is considered, the input gain from conferences is beneficial and tips and suggestions from colleagues and friends are valuable. As the amount of components out there is enormous it may be difficult for the developers to gain an overview of the available content. Therefore the input gained from personal and social resources are extremely profitable to support the selection of components.

8.2 Formal methods not used

An interesting question to discuss is why none of the many formal selection methods that are presented in research literature is used. There may be several explanations for this. One could be that the companies do not even know that such formal methods exist. Working in a company with tight schedules and frequent deliveries may neglect keeping track of research. The only way I see this kind of research may be perceived by the developers is if they happen to be at a conference where a method like this is presented. A second reason that is also noted in (Li, Conradi, Bunse, Marco, Slyngstad, & Morisio, 2008), is that many of the formal methods lack an validity check in form of a case study or reports of successful adaption. Without such an extensive validation, the developers and the companies may not dare to test the theories in practice in fear of failure. A third reason may be that the companies see the methods as too time consuming and bureaucratic, and therefore, maybe, the savings in form of using OSS instead of develop it from scratch decrease to a point where the gain is minimal.

8.3 Lack of an solution to aid the selection process

Several of the interviewees missed a website or solution that could gather experiences with OSS components and provide useful resources to aid the selection process. Some exist like for example Ohloh, but they are not satisfactory. Ohloh lacks e.g. a categorization of the OSS projects, in such a way that if a developer wants a GUI component, he could consult the GUI category. A wiki to address the lack of an informative website has been proposed by researchers at Technical University Of Catalunya (UPC) and Norwegian University of Science and Technology (NTNU) as discussed in the paper from 2007(Ayala, Sørensen, Conradi, Franch, & Li, 2007). This OTS-Wiki has three high level goals listed in Table 13.

ID	High level goals
1	“Fostering an OTS Community and Incremental Population of Content.”
2	“Federating Actual Efforts for Locating and Selecting OTS Components.”
3	“Enabling Systematic Support for Selecting and Evaluating OTS Components. “

Table 13 High level goals of the OTS-Wiki

The first goal implies that they wish the OTS-Wiki will encourage and help OTS developers to share knowledge in an incremental way, taking one step at a time. This way the amount of experiences and knowledge will gradually increase as the developers add this kind of information to the wiki. The next goal is that the authors hope that the wiki could help the developers in the identification and selection process of components by letting the users add hyperlinks to helpful existing web resources to aid the selection process. Examples here may be sites like Ohloh, or tools like "Code Conjurer" which will be discussed in the next paragraph. The third and last goal implies that the OTS-Wiki will aid the developer in the evaluation of components by having a structured set of information available. The DesCOTS system (Grau, Carvallo, Franch, & Quer, 2004) will be integrated into the wiki, to support this evaluation process.

A tool that is available for free download in a beta-version is "Code Conjurer"[1]. This is an Eclipse plug-in that could be used to aid the developer in finding code-snippets and components while developing in a test-driven development fashion (Hummel, Janjic, & Atkinson, 2008). As the developer writes his tests, the plug-in automatically searches for fitting components in the Merobase[2] server, tests them in a virtual environment, and returns the filtered results of snippets/component to the user. If the developer wants to use it, Code Conjurer may automatically resolve any dependencies and integrate it into the project. This tool may ease the overall development efforts for a Java based project, and maybe even find components that may not have been identified if the tool was not used.

8.4 Poor OSS documentation

One of the findings of this research that do not exactly answer any of the research questions, but that still interesting to note, is that many OSS projects have poor documentation. Many of the respondents keenly stated this during the interviews. In a research conducted by interviewing several managers in firms in the European secondary software sector Morgan and Finnegan found that documentation in many OSS projects was often outdated or may have died in development(Morgan & Finnegan, 2007). Capiluppi, Lago and Moriso reports that poor documentation is usually the case for small projects, but that the documentation gets better as the project grows (Capiluppi, Lago, & Morisio, 2003). Indeed this may be the case, considering that the WiX project which one of the respondent mentioned as an example of a well documented OSS project. The WiX project has become a large project with a large company backing up the project, namely Microsoft.

But why is the documentation in many OSS projects bad? One answer to this may be that developers do not like producing documents, and since many OSS projects are entirely based on voluntary work, the documentation is seen upon as a boring thing that no one wants to perform. Developers may rather want to produce new functionality or fix important bugs than do less interesting things like documentation. Additionally, as

[1] http://codeconjurer.sourceforge.net
[2] http://www.merobase.com/

also stated by Bonaccorsi and Rossi, since work often is not assigned as it is in a more commercial setting the documentation is simply overlooked (Bonaccorsi & Rossi, 2003). A third explanation for the poor documentation may be that the communities see their project as intuitive and thus the documentation is not necessary. Working with the software often, maybe each day, could result in such a view of the software and that may not be exactly true.

9 Limitation and validity

Since this study mainly is focused on Norwegian software companies drawn conclusions from the discussion of the results might not necessary be correct in every case. As mentioned in Research design and context the Norwegian software industry has a flat structure, and may be more fitted to a bottom-up open source adoption process than for example larger American companies with a more hierarchal structure. This been said there exists companies in Norway that have a more hierarchal structure, and one of the respondents in this research mentioned that they needed a final go from someone with higher rank.

Next the amount of data collected may not be of an adequate size. Only 5 companies were interviewed during this project assignment, thus a larger amount of interviews should have been conducted to yield more results to validate the theories presented. In addition a limitation is that of what could be called convenient sampling. Which means that the companies selected to participate in this study was not chosen completely at random but rather as what was most befitting. Thus, to avoid travelling and having to conduct telephone interviews, the most fitting companies were those who had a developing unit in Trondheim. Another thing that may decrease the overall generality of this study is that four of the projects investigated were very similar. All these four projects were java projects, web-based and used the Spring framework. Further two of the projects used the commons libraries and the other two used Hibernate. Since four of the five projects investigated were pretty similar it could be difficult to generalize the concepts found.

Finally, during the interview I should maybe have been more alert on the mix up between the OTS and the COTS term. Many of the interviewees was not familiar with the term OTS and even though I had a quick intro explaining that this term covered both COTS and OSS, a few of the respondents was confused and answered the question considering OTS components with only COTS components in mind. Maybe a more thorough intro should be given in advance of the interview or maybe it may have been advantageous to provide the interviewee with a definition-list.

10 Conclusions and Future Work

The findings of this study suggest that the identification process is twofold in terms of a proactive and a reactive part. In the proactive part companies maintain a general awareness of OSS components out there with aid from books, websites, forums, conferences and through their personal social network. The reactive part is formed by an instant need of a solution to a specific problem. This may be a problem that the company does not have an immediate idea on how to solve or it may be outside their general knowledge domain. Google is important source for this reactive part of the identification. The evaluation seems to be of a more ad-hoc manner instead of following the formal methods presented in research literature. Companies tend to put much emphasis on downloading the component and construct an easy prototype. Prototyping together with an assessment of available documentation and component maturity seems to be important for the evaluation. The choice of component is often of a first fit manner and especially in case of previous positive experience with a component. To aid this whole selection process companies use both Internet resources (blogs, RSS feeds etc.) and personal and social resources (conferences, books, recommendations etc.)

My findings have some consequences for various groups of people. Firstly developers and integrators may profit of being more aware of how they do the final choice of components. They often do a first-fit choice of OSS component and thus might not choose the best component. Further the study shows the developers that some solutions to aid the selection process do exist, such as Ohloh and Code Conjurer. Secondly this study should indicate to the OSS communities that an increased focus on documentation is necessary to be able to grow as a project and get more download and hence more users. The effort to produce an easy FAQ or Quick-guide is not that large and may prove remarkably profitable for the community. Additionally putting in a small effort to check the forums for questions and answer them within reasonable time could increase the confidence of possible adaptors. Finally, researchers that study and research on OSS related issues should accept that the formal methods most likely will not be used, if they is to have any chance of being adopted the methods has to be thoroughly tested through case-studies. Therefore it will be benefitting that researchers rather shifts their focus and try to research on tools, methods or solutions that may aid the more ad-hoc selection process that seems to be present in the software industry today.

To validate the findings in this research a broader and more diverse range of respondents should be studied either in the form of more interviews or by a questionnaire, this is subject to future research. Also, it could have been interesting to ask the developers if they know that several formal methods exist, and if so, why do they not use them? Developing a knowledge database, a website or a tool to aid the selection process is also options for future work, as developers have expressed their desire for a central place to find experiences and support the selection of OSS components. Developing of the OTS-Wiki presented by (Ayala, Sørensen, Conradi, Franch, & Li, 2007) is also a possible alternative for future work.

Glossary

FOSS	Free and Open Source Software
OTS	Of-The Shelf
OSS	Open Source Software
COTS	Commercial-Of-The-Shelf
FLOSS	Free/Libre/Open Source Software
OSI	Open Source Initiative
FSF	Free Software Foundation
GPL	General Public License
BSD	Berkeley Software Distribution
LGPL	Lesser-General Public License (former Library General Public License)
Copyleft license	License type that ensures that the licensed software has the same freedoms in the modified versions as it had in the original version
CBSD	Component-based software development

References

Aivosto Oy. (n.d.). *Chidamber & Kemerer object-oriented metrics suite*. Retrieved October 10, 2008, from http://www.aivosto.com/project/help/pm-oo-ck.html

Atos Origin. (2006, April). *Method for Qualification and Selection of Open Source software(QSOS), version 1.6*. Retrieved October 9, 2008, from http://qsos.org/download/qsos-1.6-en.pdf

Ayala, C. P., Sørensen, C.-F., Conradi, R., Franch, X., & Li, J. (2007). Open Source Collaboration for Fostering . *IFIP Vol 234, Open Source Development, Adoption and Innovation* (pp. 17-30). Springer.

Basili, V. R. (2001). COTS-based Systems Top 10 List. *Computer*, (pp. 91-95).

Bonaccorsi, A., & Rossi, C. (2003). Why Open Source software can succeed. *Research Policy 32* (pp. 1243-1258). Elsevier Science.

BRR 2005 - RFC 1. (2005). *Business Readiness Rating for Open Source - A Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software*. Retrieved October 9, 2008, from http://www.openbrr.org

Capiluppi, A., Lago, P., & Morisio, M. (2003). Characteristics of Open Source Projects. *Proceedings of the Seventh European Conference On Software Maintenance And Reengineering (CSMR'03).* IEEE - Computer Society.

Chen, W., Li, J., Ma, J., Conradi, R., Ji, J., & Liu, C. (2008). An Empirical Study on Software Development with Open Source Components in the Chinese Software Industry. *Software Process: Improvment and Practice*, (pp. 89-100).

Cruz, D., Wieland, T., & Ziegler, A. (2006). Evaluation Criteria for Free/Open Source Software Products Based on Project Analysis. *Sofware Process Improvment And Practice* (pp. 107-122). John Wiley & Sons, Ltd.

Fitzgerald, B. (2006). The Transformation of Open Source Software Software. (pp. 587-598). MIS Quarterly Vol. 30 No. 3.

Free Software Foundation. (2008, December 11). *Free Software Foundation*. Retrieved December 15, 2008, from Free Software Foundation Files Suit Against Cisco For GPL Violations: http://www.fsf.org/news/2008-12-cisco-suit

Free Software Foundation, Inc. (n.d.). *GNU Operating System*. Retrieved August 27, 2008, from http://www.gnu.org/philosophy/free-sw.html

Golden, B. (2005). Making Open Source Ready for the Enterprise: The Open Source Maturity Model. *Succeeding with Open Source.* Addision Wesley.

Golden, B. (2005). *Succeeding with Open Source.* Addison Wesley.

Grau, G., Carvallo, J., Franch, X., & Quer, C. (2004). DesCOTS: A Software System for Selecting COTS Components. *30th EUROMICRO Conference (EUROMICRO'04)* (ss. 118-126). IEEE Computer Society.

Hummel, O., Janjic, W., & Atkinson, C. (2008). Code Conjurer: Pulling Reusable Software out of Thin Air. IEEE Software.

Jaaksi, A. (2007). Experiences on Product Development with Open Source Software. *IFIP International Federation for Information Processing* , pp. 85-96.

Li, J., Conradi, R., Bunse, C., Marco, T., Slyngstad, O. P., & Morisio, M. (2008). Development with Off-The-Shelf Components: 10 Facts. *IEEE Software* .

Merilinna, J., & Matinlassi, M. (2006). State of the art and practice of open source component integration. IEEE Computer Society.

Microsoft Corporation. (2008). *Windows Installer XML (WiX) toolset*. Retrieved November 27, 2008, from http://wix.sourceforge.net/

Miles, M. B., & Hauberman, A. (1994). *Qualitative data analysis: An expanded sourcebook (2nd ed.).* Newbury Park, CA: Sage.

Mohamed, A., Ruhe, G., & Eberlein, A. (2007). COTS Selection: Past,Present, and Future. *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07).*

Morgan, L., & Finnegan, P. (2007). Benefits and Drawbacks of Open Source Software: An Exploratory Study of Secondary Software Firms. *IFIP Vol 234, Open Source Development, Adoption and Innovation* (pp. 307-312). Limirick, Ireland: 307-312.

Myers, M. D., & Newman, M. (2006). The qualitative interview in IS research: Examining the craft. *Elsevier - Information and Organization.*

Navica Inc. (2008). *Navica*. Retrieved October 13, 2008, from http://navicasoft.com

Oates, B. J. (2006). Researching Information Systems and Computing. In B. J. Oates, *Researching Information Systems and Computing* (pp. 186-199). Sage Publications Ltd.

Open Source Initiative. (2006, July 7). *The Open Source Definition*. Retrieved September 12, 2008, from http://www.opensource.org/docs/osd

Raymond, E. S. (2001). The cathedral and the bazaar. Sebastopol, CA: O`Reilly.

Red Hat Middleware, L. (2006). *hibernate.org*. Retrieved December 4, 2008, from http://www.hibernate.org/

Robson, C. (2002). *Real world research a resource for social scientists and practitioner researchers.* Oxford: Blackwell.

Rusin, Z. (n.d.). *Open Source Licenses*. Retrieved September 12, 2008, from http://developer.kde.org/documentation/licensing/licenses_summary.html

Samoladas, I., Gousios, G., Spinellis, D., & Stamelos, I. (2008). The SQO-OSS Quality Model: Measurment Based Open Source Software Evaluation. *IFIP Volume 275; Open Source Development, Communities and Quality* (pp. 237-248). Boston Springer.

Scacchi, W. (2007). Free/Open Source Software Development: Recent Research Results and Emerging Opportunities. *ESEC/FSE '07* (pp. 459-468). Cavtat, Canada: ACM.

Soini, J., Keto, H., & Makinen, T. (2006). An Approach to Monitoring the Success Factors in Software Business in Small and Medium Size Software Companies. *PICMET 2006 Proceedings* (pp. 2801-2808). Istanbul, Turkey: PICMET.

SpringSource. (2008). *SpringSource Community*. Retrieved November 27, 2008, from http://www.springframework.org/

StatBank Norway, 2008. (n.d.). *SSB*. Retrieved September 30, 2008, from http://statbank.ssb.no/statistikkbanken

Taibi, D., Lavazza, L., & Morasca, S. (2007). OpenBQR: a framework for the assesment of OSS. *IFIP, Volume 234, Open Source Development, Adoption and Innovation* (pp. 173-186). Boston Springer.

Testwell Oy . (2007). *Verisoft Technology*. Retrieved October 10, 2008, from Measurement of Halstead Metrics with Testwell CMT++ and CMTJava (Complexity Measures Tool): http://www.verifysoft.com/en_halstead_metrics.html

Umarji, M., Elliott, S. S., & Lopes, C. (2008). Archetypal Internet-Scale Source Code Searching. *IFIP Vol.275; Open Source Development, Communities and Quality* (pp. 257-263). Boston: Springer.

Vigder, M. R., Gentleman, W., & Dean, J. (1996). COTS Software Integration: State of the art. NRC, 39198.

Appendix A – OSS component information

Table 14 lists the OSS components mentioned by the interviewees and their respective web-sites.

OSS Component	Web site
OpenOffice SDK	http://download.openoffice.org/3.0.0/sdk.html
WIX	http://wix.sourceforge.net/
GCC	http://gcc.gnu.org/
nAnt	http://nant.sourceforge.net/
nUnit	http://www.nunit.org/index.php
Nullsoft Scriptable Install System	http://nsis.sourceforge.net/Main_Page
Spring	http://www.springframework.org/
div commons libraries	http://commons.apache.org/
AOP Alliance	http://aopalliance.sourceforge.net/
Spring Web Services	http://static.springsource.org/spring-ws/sites/1.5/
Hibernate	http://www.hibernate.org/
Lucene	http://lucene.apache.org/java/docs/
log4j	http://logging.apache.org/log4j/
MySQL	http://www.mysql.com/
Google Web toolkit	http://code.google.com/webtoolkit/
JQuery	http://jquery.com/
Postgres	http://www.postgresql.org/
Java Server Faces	http://java.sun.com/javaee/javaserverfaces/
Facelets	https://facelets.dev.java.net/
RichFaces	http://www.jboss.org/jbossrichfaces/
JasperReports	http://jasperforge.org/plugins/project/project_home.php?group_id=102
Myfaces Tomahawk	http://myfaces.apache.org/tomahawk/index.html

Table 14: Appendix - OSS component information

Appendix B – Interview guide and transcripts

The interview guide is also available at the following URL:

http://folk.ntnu.no/ketilsan/project08

The full transcripts are available, in Norwegian, at the following URL:

http://folk.ntnu.no/ketilsan/project08/transcripts

Or consult Table 15 for a specific interview transcript.

Company	URL
Alpha	http://folk.ntnu.no/ketilsan/project08/transcripts/Intervju1.docx or http://folk.ntnu.no/ketilsan/project08/transcripts/Intervju1.pdf
Beta	http://folk.ntnu.no/ketilsan/project08/transcripts/Intervju2.docx or http://folk.ntnu.no/ketilsan/project08/transcripts/Intervju2.pdf
Gamma	http://folk.ntnu.no/ketilsan/project08/transcripts/Intervju3.docx or http://folk.ntnu.no/ketilsan/project08/transcripts/Intervju3.pdf
Delta	http://folk.ntnu.no/ketilsan/project08/transcripts/Intervju4.docx or http://folk.ntnu.no/ketilsan/project08/transcripts/Intervju4.pdf
Epsilon	http://folk.ntnu.no/ketilsan/project08/transcripts/Intervju5.docx or http://folk.ntnu.no/ketilsan/project08/transcripts/Intervju5.pdf

Table 15: Appendix - URL for transcripts