A Comparison of Motivation and Openness in Hybrid Open Source and Open Source Software Projects

Steinar Hagen

Master of Science in Informatics
Submission date: February 2011
Supervisor: Reidar Conradi, IDI
Co-supervisor: Daniela Soares Cruzes, IDI
Abstract

**Context:** Open Source software development is now becoming used for commercial software development, where software companies manage open source projects that are open for everyone who wants to contribute. This is called Hybrid Open Source Software since it stands with one foot in the Open Source camp and one foot in the commercial development camp. Since hybrid open source software is a new phenomenon, it is interesting to compare both the projects and its contributors to more traditional open source projects.

**Objective:** Based on two research questions (RQ1-RQ2) this thesis aims to find out what motivates the contributors in hybrid open source projects and how democratic company maintained open source projects are.

**RQ1:** What are the differences in motivation for the various types of open source projects?

**RQ2:** How open are hybrid projects compared to other OSS projects?

**Method:** Four software projects have been closely investigated in order to answer RQ1 and RQ2. These have mostly been compared in pairs of one hybrid and one open source project. The following projects have been investigated (1) Qt framework (hybrid) (2) KDElibs framework and (3) Asterisk PBX software and (4) FreePBX PBX Software. Research has been conducted by structuring and examine comments in bug reports and feature requests on the projects web sites.

**Contributions:** Based on the information collected and analyzed the following main contributions have been made:

**C1:** Added to previous research on hybrid open source projects

**C2:** Collected and presented evidences on differences between hybrid open source projects

**C3:** Evidences on different treatment of internal and external contributors in hybrid projects

**Conclusions:** Companies organizing software development projects as hybrid open source projects does without doubt benefit from doing so. Large numbers of external contributors stand for most of the initiated bug reports and suggestion of features, and can under the right conditions make it possible with rapid releases of well-crafted software.
Preface and Acknowledgements

This thesis is the final result of IT3901, Master thesis in Informatics at the Department of Computer and Information Science (IDI) at Norwegian University of Science and Technology (NTNU).

This thesis has been created under the supervision of main supervisor Professor Reidar Conradi (IDI) and co-advisor Post Doctoral Fellow Daniela Soares Cruzes (IDI).

Acknowledgements

This thesis is based on the effort of several people;
First, I would like to thank my supervisors, Professor Reidar Conradi and co-supervisor Post Doctoral Fellow Daniela Soares Cruzes, both at the Department of computer science and for their continuous support through a continuously changing context and problem description. I also would like to thank everyone at Nokia’s Qt department in Oslo, especially Community Manager Knut Yrvin for giving his thoughts on the problem description of the thesis in the beginning and for sharing his thoughts on how the community around Qt and hybrid projects will evolve in the future. This was essential for establishing a problem description for the thesis.
Monika Olsen also deserves a great thank for proofreading my somewhat broken English in the very last stages of the writing process.

Least, but not last I would like to thank my parents for their lifelong consideration and support.

Oslo, February 1, 2011

Steinar Hagen
# Table of Contents

## Contents

Abstract .......................................................................................................................... 1
Preface and Acknowledgements .................................................................................... 3
Table of Contents .......................................................................................................... 5
List of Tables ............................................................................................................... 8
List of Figures ............................................................................................................ 9
Acronyms and Abbreviations ....................................................................................... 10
Chapter 1 .................................................................................................................. 11
  1.1 Motivation ............................................................................................................. 11
  1.2 Rationale and background .................................................................................. 11
  1.2 Problem Statement .............................................................................................. 12
  1.3 Research contributions ....................................................................................... 12
  1.4 Thesis Structure .................................................................................................. 12
Chapter 2 .................................................................................................................. 14
State of the art ............................................................................................................ 14
  2.1 Open source Software .......................................................................................... 14
    2.1.1 Hacker Culture .............................................................................................. 14
    2.1.2 GNU/Free Software Foundation and Open Source Initiative .................... 15
    2.1.3 Proprietary goes open .................................................................................. 16
  2.2 Motivation ........................................................................................................... 17
    2.2.1 Intrinsic and extrinsic motivation ................................................................. 17
    2.2.2 Theory X and Y ........................................................................................... 17
    2.2.3 Two-factor theory ....................................................................................... 18
    2.2.4 Maslow’s hierarchy of needs ....................................................................... 19
  2.3 Motivation in software engineering ..................................................................... 19
    2.3.1 Motivation in Open Source Software (OSS) projects ................................ 20
  2.5 Software engineering ......................................................................................... 21
    2.5.1 Traditional software engineering ................................................................. 21
    2.5.2 Off-The-Shelf (OTS) .................................................................................. 22
    2.5.3 Distributed software engineering ............................................................... 22
  2.6 Software engineering in open source development .......................................... 22
2.6.1 Parallel development .................................................................................. 23
2.6.2 Communities ............................................................................................... 23
2.6.3 Independent peer review ........................................................................... 23
2.6.4 Prompt feedback ......................................................................................... 23
2.6.5 Highly talented and motivated developers ................................................... 23
2.6.6 Involvement of users ................................................................................. 23
2.6.7 Rapid release schedule .............................................................................. 23
2.7 Intellectual Property Rights IPR ..................................................................... 24
  2.7.1 What is copyright? ..................................................................................... 24
  2.7.2 Software licenses ....................................................................................... 24
2.8. Software patents ........................................................................................... 26
  2.8.1 Criticism of software patents ................................................................... 26
2.9 Openness in software engineering .................................................................. 26
Chapter 3 ............................................................................................................. 28
Research design ................................................................................................... 28
  3.1 Context and motivation ................................................................................ 28
  3.2 Research questions ...................................................................................... 29
    3.3.1 RQ1 What are the differences in motivation for the various types of open source projects? ............................................................................. 29
    3.3.2 RQ2 How open are hybrid projects compared to other OSS projects? .......................................................... 30
Chapter 4 ............................................................................................................. 33
Results and main findings .................................................................................... 33
  4.1 Project presentations ..................................................................................... 33
    4.1.1 Qt ........................................................................................................... 33
    4.1.2 The KDElibs community ........................................................................ 33
    4.1.5 Asterisk .................................................................................................... 34
    4.1.6. Free PBX ............................................................................................... 34
  4.2 Main findings ................................................................................................. 35
    4.2.1 Findings from RQ1 ................................................................................ 35
Chapter 5 ............................................................................................................. 45
Evaluation and Discussion .................................................................................. 45
  5.1 Answering the Research Questions ............................................................... 45
    5.1.1 Answers RQ1 .......................................................................................... 45
    5.1.2 Answers to RQ2 ...................................................................................... 47
  5.2 Limitations, validity and reliability ............................................................... 48
Chapter 6 ............................................................................................................. 49
Conclusions and Further Work .......................................................................... 49
  6.1 Contributions ................................................................................................ 49
6.2 Further work ........................................................................................................ 50
Glossary ...................................................................................................................... 51
References .................................................................................................................. 52
## Acronyms and Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COTS</td>
<td>Commercial Off-The-Shelf</td>
</tr>
<tr>
<td>FLOSS</td>
<td>Free/Libre/Open Source Software</td>
</tr>
<tr>
<td>NTNU</td>
<td>Norwegian University of Science and Technology</td>
</tr>
<tr>
<td>OSS</td>
<td>Open Source Software</td>
</tr>
<tr>
<td>OTS</td>
<td>Off-The-Shelf</td>
</tr>
<tr>
<td>RQ</td>
<td>Research Question</td>
</tr>
<tr>
<td>IDI</td>
<td>Department of Computer and Information Science</td>
</tr>
<tr>
<td>IPR</td>
<td>Intellectual Property Rights</td>
</tr>
<tr>
<td>GNU</td>
<td>Gnu’s Not Unix</td>
</tr>
<tr>
<td>GPL</td>
<td>General Public License</td>
</tr>
<tr>
<td>FSF</td>
<td>Free Software Foundation</td>
</tr>
<tr>
<td>DRM</td>
<td>Digital Rights Management</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Source Initiative</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter gives a short introduction to the research undertaken, problems to be addressed, previous research within the field and the structure of the thesis.

1.1 Motivation

Open Source Software (OSS) has forever changed the way we look at software; it has changed the software industry and given birth to several new business models. Open source Software is today used by everyone, from individuals to large corporations, and in everything, from cell phones to robots for space exploration\(^1\) (Norris 2004).

The software industry has shifted its focus to Open Source Software business models. The use of Open Source Software in the industry is increasing, and many new software products are released as open source, including Sun Microsystems’ decision to turn its entire product portfolio, including the Java programming language and the Solaris Operating system, to open source\(^1\).

Several public and governmental organizations have taken a public viewpoint on supporting and choosing Open Source systems where available, this also includes Norway’s current government\(^2\).

A more recent evolution within the software development biotope is Open Source projects managed and initiated by the software industry. Organized as more traditional Open Source projects, these projects encourage developers and other contributors from outside the organization to contribute. This is what I have chosen to call hybrid open source projects throughout this thesis.

The strong interest in open source has led to several definitions on Open Source and free software, and similarly misunderstandings on what open source is, should be, and is common. The Free Software Foundation, founded by Richard Stallman, argues that changes made to already open software should be released as open software. The Open Source Initiative practices a more liberal viewpoint where open source software can be used in commercial software.

1.2 Rationale and background

With the widespread use of Open Source Software everywhere, and the fact that almost everyone is using Open Source software, either as a deliberate action or simply by using systems that rely on Open Source components. It is interesting to study why and how OSS is developed. Why do commercial companies spend money and workforce on Open Source?

\(^1\) Sun opens Java, press release, 2006:
Why do persons freely distribute their code? However, the large majority of users who download Open Source Software are “freeriders”, persons using a product without contributing to its further development as described by Hippel and von Krogh, 2003. It is therefore interesting to study who are the developers and contributors to Open Source projects, and what motivates them to make contributions.

While several studies have been conducted on motivation among software developers in software companies and among core developers of Open Source Software, motivation in hybrid projects are still a field with little knowledge.

What is known about hybrid projects is that there is a division between the internal and external developers. The internal developers often have additional communication platforms (mailing list, repositories, IRC channels) along with the communication channels open for everyone. Some companies, like Qt Software, are aware of this segregation, and want to look into what can be done in order to reduce this division.

1.2 Problem Statement

The problem description for this thesis is to find whether the motivational factors for involvement in a hybrid project are different from involvement in an open source project, and if there are any differences in decision making and openness in hybrid projects compared to open source projects.

Two research questions are extracted from this problem statement. Further discussion around these is placed in chapter 3, Research Design.

**RQ 1**

What are the differences in motivation for the various types of open source projects?

**RQ 2**

How open are hybrid projects compared to other OSS projects?

Table 1.1 Research questions

1.3 Research contributions

The contributions from this thesis are study of a two pairs with hybrid open source projects and open source projects.

Also, the state-of-the art chapter presents a theoretical basis from research inside OSS communities, hybrid projects and from commercial software development.

1.4 Thesis Structure

This thesis is divided into six chapters, including this introduction.

**Chapter one** is the introduction.
Chapter two contains background and state-of-the-art from the field of subject. First a historical walk-through of what is known as Open Source software today is given, then the most common theories of motivation is presented. Findings from studies on motivation of developers in OSS and commercial projects are presented. This chapter also gives an introduction to software engineering and how software engineering is done traditionally and in OSS project. Finally, the chapter gives a short presentation of intellectual property rights (IPR) and software patents.

Chapter three gives an introduction to the tools and methods being used to collect data.

Chapter four presents the results from the research undertaken.

Chapter five consists of an evaluation and discussion of the results presented in chapter four.

Chapter six formalizes the results presented and discussed in chapter four and five. The discussion is followed by a section on future and further work that should be conducted within this field.
Chapter 2

State of the art

This chapter presents the most common definitions and views on Open Source and free software. Then, it moves on to an overview of the origin and history of open source software up until today, and examples of some well-known open source software are given. Second part of this chapter covers the most common theories about motivation, and some of the research done when it comes to motivation of software developers. The final part of the chapter discusses the different licensing types that are common to release software under, and a short introduction to software licences, intellectual property rights (IPR) and problems related to such in free software.

2.1 Open source Software

Open source software has become an everyday concept, and has found its way into colloquially for a diversity of definitions on what open source actually is. The term open source is often mixed with similar terms as free software and freeware, although these terms are not similar. Common for all terms is that open and free software is free of charge to use, and that the code is available for everyone.

The Free Software Foundation enlists this among its definition of free software "Free software is software that gives you the user the freedom to share, study and modify it. We call this free software because the user is free." The Free Software Foundation (FSF) focuses on the ethical and moral aspects on the use of software.

The Open Source Initiative on the other hand, have a wider definition on open source software where the rights to redistribute the software are the central part.

2.1.1 Hacker Culture

Computer software and computer programming as we know it today evolved in the early 1950s, mainly in universities and research institutions. Software development was a tedious process, and it was hard to get a program to work correctly and efficient. Hardware was extremely expensive, and limited, so that users needed to make sure their code was functioning before using valuable run time on a computer. As a consequence of this, source code was shared among users, as well as tips and techniques for optimizing programs. In many cases the software itself was just a tool for creating statistics, graphs or similar.

Software was in many cases developed by scientists and researchers who found it natural to share and exchange their software along with their research results. This practise derives from a long tradition of sharing results and methods among universities and research institutions. Research results are often viewed upon as a public good, and influences from this way of thinking have probably influenced the early programmers.

---

ii http://www.fsf.org/about/what-is-free-software

iii http://www.opensource.org/docs/osd
In 1953, the aircraft manufacturers Lockheed and Douglas, together with North American Aviation, formed the Project for Advancement of Coding Techniques (PACT). This was probably the first initiative to share code across company borders Leonard (2000).

This culture of openness among the earliest computer programmers has later been given the name Hacker culture. A hacker was originally a person that used an axe to create a subject out of wood, but is nowadays associated with crimes related to theft of information from computer systems. The term hacker in software terms originates from the Tech Model Railroad Club (TMRC) at MIT, and were later associated with the MIT Artificial Laboratory, and the Homebrew Computer Club in Silicon Valley.

2.1.2 GNU/Free Software Foundation and Open Source Initiative

In 1984 Richard Stallman, a young software programmer at MIT, quit his job at MIT’s artificial intelligence laboratory to work full time writing free software. Stallman was dissatisfied with a change in the software industry, where most of the previously open and modifiable code went closed source, and where licenses restricted the modification of software to a minimum.

Two years earlier, Stallman had released the GNU project aimed to create an entirely free and open operating system called GNU compatible with existing Unix operating systems iv. From this the name which stands for Gnu's Not Unix derived. This work was even more formalized with the GNU manifesto v in 1985 and the General Public License (GPL) in 1989. The GNU manifesto is a description of Stallman’s thoughts around free software, some of the GNU projects that existed in 1985. It does also include examples of how programmers can make a living without selling software itself.

Along with the GNU manifesto Stallman also established the Free software Foundation, a non-profit organization established to “promote computer user freedom and to defend the rights of all free software users” vi. This organization played a major role in developing applications for GNU, but in later years the focus has shifted towards advocating for free software, and campaigning against Digital Rights Management (DRM), cloud-based network storage, and campaigning for open document formats and hardware designed for free software vii.

However, the probably most notable contribution to the open source environment from the FSF is the General Public License. It is the most used license on free software. Of more than 43,000 projects on the website freshmeat.net, 21,000 are licensed under GPL viii. In GPL, Stallman introduced the golden rule stating: “I consider that the Golden Rule requires that if I like a program I must share it with other people who like it” and thereby requiring users who write changes in GNU software to publish them.

---

iv https://groups.google.com/group/net.unix-wizards/msg/4dadd63a976019d7?dmode=source&hl=en
v http://www.gnu.org/gnu/manifesto.html
vi http://www.fsf.org/about/
vii http://www.fsf.org/campaigns
viii http://freshmeat.net/search?page=1&q=&submit=Search&with=&without=
The GPL license has been criticized for its viral nature, since the license states that code containing GPL licensed code has to be publically available. It is therefore argued that computer programs created for commercial sale should avoid using the GPL license\textsuperscript{ix}. The licensing also makes it more difficult to release GPL licensed software with two licenses (dual licensing) as many provider of hybrid open source software does. The provisions in GPL have perhaps also been a leading factor for the plethora of different licenses that open source software is released under.

In 1998 the Open Source Initiative (OSI) was founded by Bruce Perens and Eric S. Raymond trigged by Netscape’s release of Netspace Navigator. OSI created a definition for open source software, based on Debian Free Software Guidelines\textsuperscript{x}. OSIs definition consists of ten points that must be fulfilled in the license terms in order to call a program “open source”, among these are the principle of free redistribution of the code and allowance of modification and derivate works of the original code\textsuperscript{xi}.

The open source definition by OSI has been criticized by Richard Stallman and the Free Software Foundation for not involving the freedom of the users. In his public letter “Why Open Source misses the Point”, Stallman argues that “Open source is a development methodology; free software is a social movement. For the free software movement, free software is an ethical imperative, because only free software respects the users' freedom.”\textsuperscript{xii}

The definition most users mean in terms of free or open source software is relatively closely to the one Open Source Initiative uses.

2.1.3 Proprietary goes open

There are several examples of software that initially were published as closed source and with proprietary licenses, have been released as open source with various results, and for various motives.

In early 1998, Netscape Communication donated large parts of source code of their product Netscape Communicator as open source. This decision is said to be motivated by Raymond’s essay “The Cathedral and the Bazaar”\textsuperscript{iv} (Raymond 1998). This release later gave birth to the popular Mozilla Firefox web browser and the Thunderbird email client. Along with the release of Mozilla, Netscape also took initiative to create the Open Source Initiative.

A more recent example of software that has been released as open source is the contributions made by Sun Microsystems, now a part of Oracle. Sun first released its office suite, Star Office, under the LGPL license in 2000. This was an act for reducing the dominance of Microsoft's office suite. The public released Star Office was later renamed to Open Office, and also released as GPL.

In 2005 the bulk of the source code of Solaris, Sun’s Unix clone operating system, was released under its own license. The project was discontinued in 2010 after Oracle's acquisition of Sun Microsystems, but forked (legally copied and continued) as other projects and operating systems, still under various open source licenses.

\textsuperscript{ix} http://www.freebsd.org/doc/en_US.ISO8859-1/articles/bsd GPL-ADVANTAGES

\textsuperscript{iv} http://www.opensource.org/history

\textsuperscript{x} http://www.opensource.org/docs/osd

\textsuperscript{xii} http://www.gnu.org/philosophy/open-source-misses-the-point.html
The perhaps most prominent contribution from Sun was the release of the Java platform as open source under the GPL license in late 2006. Java was developed in the early 1990s in order to run software on several platforms. Java has become one of the most popular programming languages and one of the most widely used platforms for a variety of uses.

2.2 Motivation

This section describes some of the more general theories about motivation and some of the results from research undertaken on motivation among software developers.

According to Princeton University's online dictionary, motivation is: “The psychological feature that arouses an organism to action toward a desired goal; the reason for the action; that which gives purpose and direction to behaviour” “we did not understand his motivation”; “he acted with the best of motives”

Motivation is therefore one of the forces that makes humans execute some kind of operation. Throughout history there have been made countless studies of motivation in all kinds of settings, with perhaps motivation in work as the most prominent.

2.2.1 Intrinsic and extrinsic motivation

Intrinsic motivation is described by Ryan and Deci (2000) as “doing of an activity for its inherent satisfaction rather than for some separable consequence” Ryan and Deci separates between enjoyment based and obligation/community-based intrinsic motivation. Enjoyment based motivation is, as the name suggests, activities that persons take part in because of the enjoyment of the activity itself. The outcome of doing the activity is not as important as the enjoyment of just taking part in the activity. Obligation/community intrinsic motivation is the motivation an individual can experience from participation in a group. In terms of programming, this kind of motivation can occur in the different open source development projects.

Extrinsic motivation is well described in several economic theories. It is the motivation from getting a reward as the outcome of an activity. This can be both in terms of salary for doing a task or in terms of career advancement.

2.2.2 Theory X and Y

One of the most famous theories of motivation in work environments is the X and Y theories, created by Douglas McGregor at MIT Sloan School of Management in the 1960s.

Theory X assumes that employees will avoid work as much as possible, and that they generally are lazy and dislike work. In order to ensure that the employees are working, a strong hierarchy with a narrow span of control is necessary at every level. The hierarchy is necessary because employees will avoid responsibility and ambitions as much as they can.

---

Managers in theory X must rely on threats. The only motivational factor for the employees is the salary.

Theory Y assumes that employees may have ambitions, motivation and self-control. It can further be assumed that workers enjoy their duties, and that they, given the right conditions, can execute self-direction in accomplishing tasks they are committed to. Managers of theory Y believes that people want to do well at their work, and that the satisfaction of doing a job well is a strong motivation.

McGregor stated that that the two theories are not strongly separated, but that workers in both categories are more or less common, and that managers must conduct a combined way of leadership.

<table>
<thead>
<tr>
<th>Theory X</th>
<th>Theory Y</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Workers:</strong></td>
<td><strong>Workers:</strong></td>
</tr>
<tr>
<td>Lazy</td>
<td>Enjoys work</td>
</tr>
<tr>
<td>Ineffective</td>
<td>May be motivated</td>
</tr>
<tr>
<td>No/ little ambitions</td>
<td>Creative</td>
</tr>
<tr>
<td>Avoid work</td>
<td></td>
</tr>
<tr>
<td><strong>Management:</strong></td>
<td><strong>Management:</strong></td>
</tr>
<tr>
<td>Treat</td>
<td>Trust</td>
</tr>
<tr>
<td>Coercion</td>
<td>Development of individuals</td>
</tr>
<tr>
<td></td>
<td>Open communication</td>
</tr>
</tbody>
</table>

*Table 2.1. The differences between theory Y and X workers and managerial style*

### 2.2.3 Two-factor theory

The two-factor theory was created by Frederick Herzberg in the late 1959, and is also known as Herzberg’s motivation-hygiene theory. The theory relates itself to Maslow’s theory of motivation. The theory distinguishes between motivators (challenging work, recognition, responsibility) and hygiene factors (status, job security, salary and fringe benefits).

The hygiene factors are necessary to ensure that an employee is not dissatisfied, while motivational factors are needed to inspire an employee to higher performance in his work.

<table>
<thead>
<tr>
<th>Factors for satisfaction</th>
<th>Factors for dissatisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Company policy</td>
<td>Relationship with colleagues</td>
</tr>
<tr>
<td>Achievement</td>
<td>Relationship with managers</td>
</tr>
<tr>
<td>Advancement</td>
<td>Salary</td>
</tr>
<tr>
<td>Growth</td>
<td>Security</td>
</tr>
<tr>
<td>Recognition</td>
<td>Supervision</td>
</tr>
<tr>
<td>Responsibility</td>
<td>Work conditions</td>
</tr>
<tr>
<td>Work itself</td>
<td></td>
</tr>
</tbody>
</table>

*Table 2.2. Factors for satisfaction and dissatisfaction in work environments*
2.2.4 Maslow's hierarchy of needs

This theory was proposed by the American psychologist Abraham Maslow in 1943. The theory is often portrayed graphically as a pyramid showing the five needs; physiological, safety, love and belonging, esteem and self-actualization.

- Physiological needs: These needs are required for human survival
- Safety needs: These needs could be the safety of having a job, having enough money, etc.
- Love and belonging: Needs that cover friendship, acceptance, and belonging to social groups
- Esteem: The need to have self-esteem and to be respected
- Self-actualization: This is the highest level of accomplishment a person can reach, and is often related to personal goals that may be highly individual. A person may have a goal of being one of the best in a sport, while others want to be a good parent.

Figure 2.1 Maslow's pyramid of needs

2.3 Motivation in software engineering

Several studies have been conducted among software developers in terms of what motivates them and how motivation plays a role when it comes to developing software. In 2007, Beecham et.al. did a systematic literature review of 92 papers regarding motivation in Software Engineering, and were able to draw some conclusions based on the findings in the papers.

The authors enlisted five research questions:
1. What are the characteristics of Software engineers?
2. What (de)motivates Software Engineers to be more (less) productive?

3. What are the external signs or outcomes of (de)motivated Software Engineers?

4. What aspects of Software Engineering (de)motivate Software Engineers?

5. What models of motivation exists in Software Engineering?

The first question is answered with the finding concluding that Software Engineers can be regarded as a distinct group, and that the most referred characteristics are growth oriented, introverted and with need for independence.

Question two gives a list over the most cited motivational factors. The top motivational factor is the need to identify with the task. This means a personal interest in, and understanding of the task, clear goals, having job satisfaction, and that the piece of work has a certain quality. On the other hand, the de-motivators are; poor working conditions and lack of resources.

For research question three, retention was the most common sign associated with de-motivated software engineers. Also, motivated engineers stay in their jobs longer than de-motivated ones. Some studies also showed that productivity is affected by (de)motivation. In the analysis of question four, the authors left out motivators not related to Software Engineering like salary. The findings suggested that the maintenance period of the software life cycle was the most de-motivating aspect of software engineering.

In question five, the authors looked at models to explain motivation in software engineering. The most cited model is the Job Characteristics Model (JCT). JCT states that Software Engineers will experience motivation and satisfaction if their Growth Need Strength’s (GNS) are matched by the Motivational Potential score (MPS) of the jobs they do. This model implies that Software Engineers with low GNS will be satisfied with low MPS in a job and respectively those with high GNS will need high MPS in their job.

In the final discussion, this paper states that many of the motivational factors are equal for both Software Engineering professionals and non-Software Engineering professionals.

2.3.1 Motivation in Open Source Software (OSS) projects

Open source software has traditionally been created outside the traditional company walls, and also across the regional boundaries. Therefore, it lacks some of the factors associated with traditional “in-house” development, but can be compared to projects spread across large geographical distances.

In 2005, Lakhani and Wolf studied motivational factors for a large number of developers contributing to various open source projects (Lakhani and Wolf, 2005)\(^\text{10}\). The conclusions were that enjoyment based intrinsic motivation, in other words the developers’ ability to exercise creativity, was the most important motivational factor for developers involved in open source development projects.

While working on this thesis, 18 papers about motivation in OSS development were researched. The papers were written between 2003 and 2009. This gives an indication of how much attention this field has gotten in the recent years.
The papers vary in their research methodology and research sample, but some things occur frequently; Every OSS project is different, but for larger projects, a large part of the contributors are paid for their work, meaning that they are contributing as a part of their job. There are several reasons for this; the project could be a hybrid project, like Qt, the project may be a project containing both open and closed source software, or, in the case of consultancy companies, the customer may require that the project should be publicly available.

Even if some developers receive payment for their contributions to OSS projects, many choose to contribute for other reasons. In the figure below the most common motivational factors for Open Source Software developers are enlisted.

![Motivational factors for OSS developers](image)

**Fig. 2.3 Motivational factors for OSS developers**

### 2.5 Software engineering

This section gives a brief introduction to the area of software engineering, and how it applies for hybrid open source software and traditional open source software. First a coverage of traditional software development is given, second a introduction to distributed software engineering used in corporations spread across large physical distances and working cultures and also to a great extend used in open source development processes. A section on how system requirements is gathered and defined in hybrid and open source projects is placed at the end.

#### 2.5.1 Traditional software engineering

Before looking at software engineering, a definition of software is needed. Sommerville (2001)\(^{11}\) states that software is not just code assembled to programs, but also documentation and configuration data which is needed to make a program operate correctly.
Software can roughly be divided between customized software, which is software commissioned by a particular customer, and commercial off-the-shelf (abbreviated COTS) software, which is developed by a company to be sold in an open market.

2.5.2 Off-The-Shelf (OTS)

Off-the-shelf software is, as the name indicates, a description for software that is made to be sold as a separate software package and is to be run in all kinds of organizations. Examples of OTS software are operating systems, integrated desktop editors (IDEs), spreadsheet software and graphical editing software. It is important to distinguish between off-the-shelf software, and commercial-off-the-shelf software. The word commercial does not necessarily mean software produced for purchase, while OTS can be both commercial software and open source software.

Reasons to select OTS instead of custom-made software can be the prize of the product, and the fact that it is already available for installation and customization. Drawbacks with using OTS are no, or limited influence over the development. To some extent it can be argued that open source software can be customized to the user's need, but this requires knowledge of the development tools used (programming language, software libraries etc) and good documentation.

Studies on how OTS software is selected and chosen before other OTS systems or tailored systems, shows that there is surprisingly little use of a formal selection process. Conradi et.al. (2004) found that reasons for this were limited number of OTS candidates and long-term partnerships with a specific provider of OTS software.

2.5.3 Distributed software engineering

In recent years, increases of distributed software engineering have occurred due to globalization and companies spread over large distances. Internet has to a large extent made this change possible. Where previously only larger corporations were able to conduct product development across regional borders, but now also smaller organizations can benefit distributed engineering.

2.6 Software engineering in open source development

Open source software, as all other types of software, needs software engineering, but this can be radically different for open source projects. Open source software development is described by Feller and Fitzgerald (2002) as

1. Parallel rather than linear
2. Involves large communities of globally distributed developers
3. Utilizes truly independent peer review
4. Provides prompt feedback to user and developer contributions
5. Includes the participation of highly talented, highly motivated developers
6. Includes increased levels of user involvement

7. Makes use of extremely rapid release schedules

2.6.1 Parallel development

In a parallel development project, developers work with different parts of the project at the same time. This perpetuates the modular nature of OSS products. Parallel development also makes it possible for a large number of developers to work independent on different parts of a project.

2.6.2 Communities

A community in open source terms is the collective Internet-based society with developers, other contributors and users who are collaborating on the Internet. Communities are often spread across several countries and cultures. Contributors rarely meet each other, except in various collaboration tools. One positive effect with a global team of developers is that features and requirements are viewed from an international standpoint.

2.6.3 Independent peer review

Peer review is not something unique for the open source movement, but it has been pointed from open source communities that the peer review in open source projects is more thoroughly than the one to be found in software companies where minds have a tendency to think alike [need citation, p.88 Feller & Fitzgerald]

2.6.4 Prompt feedback

Open source software relies on rapid releases, this is due to the fact that bug fixing, new features, and functionality are added by developers across different time zones, in what have been described as a constant cycle of “see bug, fix bug, bug fixed in new release” by Jørgensen 16 (2001)

2.6.5 Highly talented and motivated developers

Many of the most popular open source projects consist of a group of core developers with the other contributors under them in a strongly hierarchal structure. Contributors make their way up in the hierarchal structure by gaining respect for their contributions to the project. Highly ranked people can also attract other highly skilled people to a project. When it comes to motivation, see section [update!] for a longer discussion.

2.6.6 Involvement of users

Open source software has a reputation of listening to users. Often the users and developers are the same persons covering their personal itch, but other times, users who are not associated with a project contributes with requirements, testing of functionality etc. Often communities have websites where users can vote for which feature they would like to be implemented first from a list.

2.6.7 Rapid release schedule
Due to the previously described culture with promptness of feedback and bugfixing, and the close collaboration between contributors, most large open source projects have a tight release schedule. Most OSS projects are focused on the product and the users of the product, therefore time-to-market is considered critical.

The number of open source projects available for downloading on the Internet is countless, but many of them are developed by just one developer. Only one developer eliminates much of the need for software engineering, and such projects are in many cases developed entirely as a part-time activity or to cover a personal need (see section on motivation). Therefore many of the characteristics given to open source communities do not apply for these smaller projects.

2.7 Intellectual Property Rights IPR

In this section, a short introduction to Intellectual Property Rights, software licenses and patents on software will be given.

Before examining intellectual property (IP), it is important with an understanding of expressions. Expressions of an idea, like a written book, lyrics or computer code can be regulated by laws regarding intellectual property rights. Intellectual property laws usually state that the creator of an expression is the owner of it.

It is easy to understand the ownership of physical elements like houses, cars and books, but to understand the ownership of abstracts like code and the words within the book can be more difficult to understand. The ownership to an expression does always belong to the creator and is not an item to be bought for money.

2.7.1 What is copyright?

Copyright means literally, the right to copy. In other words, that the person or organization claiming the legal rights to an item has the right to copy it.

2.7.2 Software licenses

As mentioned previously in this chapter, software was in the early ages of computing was something that was sold together with hardware, and developed and shared among different users or developed for research purposes.

Software licenses became widespread in the 1980s due to more extensive use of workstations and an increase in the number of commercially available software. From this time the concept of floating licenses – that only a few instances of a program can be used on many computers at the same time and node locking – that the program is locked onto one computer. These are still fundamental concepts in commercial licenses. Later on, an adoption of marketing models such as “bait and hook” and discounted licenses for special user groups such as students and non-profit organizations, have been added to the family of licenses.
Open source software also has licenses, but the motivation behind the “open” type of software licences like the GPL is to protect the redistribution of the software in other products and also to give credit to the original developers.

Table 2.3 gives an overview of the most common licenses used on open source software. The table is adapted from Learner and Tirole\textsuperscript{15} (2005) and the KDE website\textsuperscript{xiv}.

<table>
<thead>
<tr>
<th>License</th>
<th>Software can be used in proprietary software</th>
<th>Redistribution of the software</th>
<th>Redistribution of code w/ changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU General Public License</td>
<td>No</td>
<td>Must be GPL compatible</td>
<td>Yes 2)</td>
</tr>
<tr>
<td>GNU Lesser General Public License</td>
<td>Yes</td>
<td>Yes 1)</td>
<td>Yes 3)</td>
</tr>
<tr>
<td>Mozilla Public License</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSD License</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Simplified BSD License</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache Software License</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes 4)</td>
</tr>
<tr>
<td>MIT License</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Qt public license</td>
<td>No</td>
<td>Only if it is open source</td>
<td>Yes</td>
</tr>
</tbody>
</table>

1. Source code for distributed LGPL libraries (if used) have to be provided
2. Only if the derivative is GNU general public license
3. Only if the derivative is GNU general public license
4. Only if the name “Apache” is not used as a part of the derivative name

\textit{Table 2.3. A comparison of the most used Open Source Software Licenses and their conditions for use.}

\textsuperscript{xiv} http://developer.kde.org/documentation/licensing/licenses_summary.html
2.8. Software patents

Software patents are an issue with great controversy, and have traditionally been avoided by open source communities. Software patents are mostly seen upon as an American phenomenon, although several countries have patent legislation which enables software to be patented.

Until the 1980s computer programs were considered by US law as mathematical algorithms, which is non-patentable, Bessen and Hunt (2004). Computer programs, code and software were protected under copyright law. After a Supreme Court decision in 1981, and various regulations and decisions during the 1980s made it easier to patent software and to obtain patents in general. According to Bessen and Hunt, about 15 percent of all patents granted per year are software patents, meaning more than 20000 software patents are filed each year.

While filers of patents are required to have detailed instructions on how the invention works, this requirement is largely eliminated for software patents by court decisions. Today, most software patents do not need to disclose source code, flow charts or detailed descriptions of the patented program (Burk and Lemley 2002 p.1162)

There is currently no official definition of what a software patent is. Allison and Lemley, 2000, states that software patents are patents where the invention is completely embodied in software, while Bessen and Hunt calls it “a logic algorithm for processing data, that is implemented via stored instructions; that is, the logic is not “hard-wired””. When it comes to filers of software patents, the study by Bessen and Hunt found that most software patents were assigned to manufacturing firms and that relatively few were assigned to firms within the software publishing industry. Many software patents are also acquired by firms known for holding large patents portfolios and pursue patents for strategic reasons.

2.8.1 Criticism of software patents

Criticism of software patents has increased in recent years. A large part of the criticism comes from professionals within the industry and from the open source movement. Many of the critics focus on the fact that not only new programs can be patented, but also principles within programming; “take away their [software developers] right to use fundamental building blocks” as Professor Donald Knuth states in a public letter to the Commissioner of Patents and Trademarks

On the other hand, advocates of software patents claims that patents often is a better way for smaller companies to protect intellectual property from being used by larger companies. There are also examples of owners of software patents who only precede legal action towards large companies using their patents.

2.9 Openness in software engineering

Open source development is often viewed as a radically new business model; the Bazaar as described by Raymond (1998). The idea of the Bazaar is a large community of developers who work together with shared values such as generalized reciprocity and meritocracy. There is a large number of contributors to open source projects, but several studies show that many

---

larger open source projects are developed by just a small number of highly active contributors. Mockus et. Al. (2000) found that the top 15 developers of the Apache web server contributed to more than 83% of the changes to the Apache source code. Ghosh and David (2003) investigated three versions of the Linux kernel and discovered that less than five developers had developed a large portion of the modules, and a significant part of the modules had only one developer.

In the same research line, Krishnamurthy (2002) analyzed the top 100 mature projects on the website Sourceforge.org and found that the median number of co-developers was only four.

Based on these findings it is reasonable to believe that open source projects are only the work of a few persons and are not as open as its name indicates. The developers of almost all open source projects can, however, be divided into core and peripheral developers. According to Terceiro and Chavez, a core developer is among one of the most active developers, who perform most of the work and, and is in general charge of the important decisions regarding the project. Peripheral contributors do not contribute as often as core developers and generally have little power over decisions. In addition to these two categories, it is useful to add a third category of contributors, those who contribute without committing code or without being “into” the project. In this category we find users who come up with suggestions or requests for features and contribute to a project with other things than code, such as translations or documentation. In this category we also find users of the projects, who find bugs while using the projects software and who is either filing bug-reports or creating patches or workarounds for the actual bug. These contributors play an important role for a mature open source project.
Chapter 3

Research design

This section gives an overview of the research questions in detail, and the motivation behind them. Then, a presentation of the methods used, and the research undertaken to answer these are given.

3.1 Context and motivation

The motivation behind undertaking this research is to suggest how internal and external developers both can equally collaborate in the same projects and to see whether it is any differences in decision making and openness in hybrid projects compared to open source projects.

The hybrid collaboration model as presented in Figure 3.1, is becoming increasingly more popular since it keeps the development costs down, and ensures rapid releases. On the other hand the challenges with the hybrid model, as all other models based on volunteers, are to keep the project interesting enough and to ensure a close cooperation and collaboration environment between the internal and external developers.

![Figure 3.1. Hybrid Open Source collaboration model](image-url)
3.2 Research questions

Studies on motivation have been conducted both in open source communities and inside companies where software is developed. When it comes to hybrid projects, most of the research covers the economic aspects and the companies’ benefits of sharing its software and opening up for external collaborators.

The aim of this thesis is to find out what makes one software project more attractive than another, and if there are any differences in motivation between internal and external developers and contributors in hybrid projects. From this problem description two research questions, RQ 1-2 have derived. These questions are summarized in the table below.

A set of hypotheses based on results from the references was created for each question. For every hypothesis, a matching research action was created to summarize the actions undertaken for answering the question.

<table>
<thead>
<tr>
<th>RQ 1</th>
<th>What are the differences in motivation for the various types of open source projects?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RQ 2</td>
<td>How open are hybrid projects compared to other OSS projects?</td>
</tr>
</tbody>
</table>

Table 3.1 Research questions

3.3.1 RQ1 What are the differences in motivation for the various types of open source projects?

Are the contributors to projects individuals who are doing this as volunteers in their spare time, or do they represent a company and are doing this as a part of their job?

Hars and Ou\(^{21}\) (2001) showed that as many as 50% of the programmers on open source projects are paid for their contributions. Studies by Koch and Schneider (2002)\(^ {22}\) and Mockus and Herbsleb (2002)\(^ {17}\) shows that as much as 80% of the code in OSS projects is committed by 20% of the developers. This somewhat disprove the myth that OSS programmers are only highly altruistic individuals who develop software to cover their «personal itch» or to learn new set of skills or gain reputation among their peers as Bonaccorsi and Rossi (2002)\(^ {23}\) shows. This question aims to find out if it is possible to link contributors to a specific company or organization based on what they write in discussion lists, from their e-mail address etc.

**Hypothesizes and research actions:**

H1.1. In all open source projects, most of the development activity is performed by a small group of core developers. In hybrid projects, these are often employees, in open source projects these individuals are often sponsored to work full-time with the project.
R1.1 For hybrid projects, lists were created of the employees with most commits. The employees were found by searching for the name in various sources on the internet. For open source projects, a list over the top 20% contributors, based on papers by Koch and Schneider and Mockus et.al. was created and these contributors labelled as core developers. The lists of people were then compared with the name of the reporter in bug reports.

H 1.2 Most of the contributors that are non-core-developers or not employed in the hybrid software organization are users of other systems who rely on the hybrid open source/open source projects examined here. When they find bugs or missing functionality as a part of their normal use and operation of the project, they submit the bug(s), and/or request changes.

R 1.2 A small sample selected from contributors with few commits were investigated to see if they were involved in other projects, and if these projects were related to the examined project in any way. These contributors were selected from contributors who had a profile on the website ohloh.net, since the majority of contributors operate with non-identifiable nicknames.

<table>
<thead>
<tr>
<th>Hypothesizes</th>
<th>Research actions</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>H1.1. Most development activity is performed by a small group of core</td>
<td>R 1.2. Create lists of the different contributors, and see how many contributions</td>
<td>“Working for free? - Motivations of Participating in Open Source Projects” Hars</td>
</tr>
<tr>
<td>developers employed or hired to work for the project.</td>
<td>to the project and bugtracker entries they have made.</td>
<td>and Ou, 2001</td>
</tr>
<tr>
<td>H1.2. Most of the registered contributors are users of other systems who</td>
<td>R 1.2 Examine if a sample of contributors with few commits are involved in other</td>
<td>“Altruistic individuals, selfish firms? The structure of motivation in Open Source software” A. Bonaccorsi and C. Rossi, 2002</td>
</tr>
<tr>
<td>rely on the hybrid open source/open source projects presented here.</td>
<td>projects, and if these projects are related to the examined project in any way.</td>
<td>“Effort, co-operation and coordination in an open source software project: GNOME” S. Koch and G. Schneider, 2002</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Two case studies of open source software development: Apache and Mozilla” A. Mockus, R.T. Fielding and J.D. Hersleb, 2002</td>
</tr>
</tbody>
</table>

Table 3.2 Research question 1

3.3.2 RQ2 How open are hybrid projects compared to other OSS projects?
Is it possible to find any differences in the openness of hybrid open source projects compared with “traditional” open source projects? The main problem here is to define openness of open source. The first task involves creating a model of hybrid and the “traditional” OSS projects and then compare them. This means finding examples of issues resolved twice, both in internal codebases/bug-trackers and in external ones for hybrid projects. Comments from contributors are also a great source for finding examples on how external contributors are treated. Also, the amount of requested features by external contributors that are approved is a good measurement for the openness. Openness in OSS software are questioned by Foshati, Moafian & Sami\textsuperscript{24} (2010), who argues that the core developers takes all critical decisions. Zhao and Deek\textsuperscript{25} (2004), argues that internal developers do most of the coding and bugfixing, and external developers submits bugs and suggest features, i.e. a semi-open process. This can be measured from of comments in the repositories.

**Hypothesizes and research actions:**

**H 2.1.** Open source projects practices a wider degree of democracy when making decisions.

**R 2.1.** The commit logs for each project were manually examined and comments were coded to create statistical material showing the response to the different suggestions proposed.

**H 2.2.** Suggestions from internal contributors are viewed as more important than suggestions from external contributors in hybrid projects. In open source projects, suggestions are more welcomed and are more likely to be accepted and implemented. Manually go through commitment logs and look for interesting comments.

**R 2.2.** Find the percentage of suggestions that is accepted. From the total number of suggestions an adequate sample is chosen and investigated. If there are some interesting comments around the suggestion this must also be included. The selection of a sample size is based on the total number of suggestions, and then scripts for excluding suggestions that were turned down.

<table>
<thead>
<tr>
<th>Hypothesizes</th>
<th>Research actions</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>H 2.1</strong> Open source projects practices a wider degree of democracy when making decisions.</td>
<td><strong>R 2.1</strong> Examine the feedback given to suggestions from contributors</td>
<td>Zhao and Deek 2004 Foshati, Moafian &amp; Sami (2010).</td>
</tr>
</tbody>
</table>
H 2.2. Suggestions from internal contributors are seemed as more important than suggestions from external in hybrid projects. In open source projects, suggestions are more greeted and is more likely to be accepted and implemented.

R 2.2. Find the percentage of suggestions that is accepted.

Table 3.3 Research question 2
Chapter 4

Results and main findings

This chapter presents the results of the research undertaken in this thesis. First a short outline is given on the four software project investigated with background information both on the product and company. Then the findings from the various projects are presented in the pairs they are examined in, but also all four of them are compared in order to show differences and similarities between the projects.

4.1 Project presentations

Below, the four selected projects are presented in the pairs that they are compared in. The projects chosen are somewhat similar in terms of number of contributors and software functions.

4.1.1 Qt

| Category: Application framework |
| License: Closed License, GNU General Public License 3, and GNU Lesser General Public License v3 |
| Contributors: 363 |
| Active since: Spring 2009. Slightly increasing code base |
| Website: http://qt.nokia.com/ |
| Product owner: Nokia |

The framework known today as Qt started up as a proprietary framework called Trolltech in the mid-1990s. In 2008 the company were acquired by the Finish mobile telephone manufacturer Nokia. The Qt framework is a cross-platform framework for developing GUI applications. Among many of its uses is the KDE desktop environment and Google Earth.

4.1.2 The KDElibs community

| Category: Application framework |
| License: GNU Lesser General Public License v3 |
| Contributors: 587 |
| Active since: Mid 2005. Almost flat curve of commits with an exception of a peak in early
KDElibs are the bases of the KDE desktop environment, used in many Linux distributions and is consisting of several software libraries based on Qt. KDElibs were chosen since they are based on Qt, have a similar number of contributors and is a generally active Open Source project. Qt, KDE and KDElibs share some of the same contributors.

4.1.5 Asterisk

Category: Software switching board
License: GNU General Public License 3.0
Contributors: 83
Active since: 2000. Constantly increasing codebase
Website: http://www.asterisk.org
Product owner: Digium Inc

Asterisk is a private branch exchange (PBX) software created by Digium in 1999. With Asterisk installed on a computer equipped with hardware for telephony it is possible to connect several telephone on one line, provide conference calls and telephone services usually found in large office building and within corporations. Asterisk is licensed under a dual license with an enterprise version of the software. There also exist several add-on software packages of the software that are sold separately and are not open source. Digium Inc manufactures and sells hardware that works with Asterisk.

4.1.6. Free PBX

Category: Software switching board
License: GNU General Public License 3.0
Contributors: 32
Active since: Late 2004. Steadily increasing code base
Website: http://www.freepbx.org/
Free PBX is covering many of the same needs as Asterisk, but focuses more on user friendliness, with several pre-configured set up options of the software. Asterisk and Free PBX are not competitors, but fills out the needs of each other. The two projects also share some of its contributors.

### 4.2 Main findings

The rest of the chapter contains diagrams and graphs answering the research questions. For a discussion on the findings, see chapter 5.

#### 4.2.1 Findings from RQ1

H1.1. In all open source projects, most of the development activity is performed by a small group of core developers. In hybrid projects, these are often employees, in open source projects these individuals are often sponsored to work full-time with the project.

**Qt and KDElibs**

*Figure 4.2 Commits from Qt (left) with internal commits in red, and commits from external contributors in blue. Similar diagram from KDElibs on right hand side, where core developers are the 20% with most commits*
Figure 4.3 Distribution of commits in Qt

Figure 4.4 Distribution of commits KDElibs

The graphs above show the number of developers (blue bars), represented on the y-axis and how many commits every developer has on the x-axis. As the graphs show, the majority
of Qt contributors have between 49-1 commits, while the KDElibs contributors are more evenly distributed, but with a peak for the contributors with just one commit.

**Asterisk and FreePBX**

![Pie charts showing distribution of commits](image)

*Figure 4.5 Commits from Asterisk (left) with internal commits in red, and commits from external contributors in blue. Similar diagram from FreePBX on right hand side, where core developers are the 20% with most commits*

![Bar chart showing distribution of commits](image)

*Figure 4.6 Distribution of commits in Asterisk*
Figure 4.7 Distribution of commits in FreePBX
Figure 4.8 Structured diagram of the different feedback given to suggested features in Qt.
Figure 4.9 Structured diagram of the different feedback given to suggested features in KDElibs.
Figure 4.10 Structured diagram of the different feedback given to suggested features in Asterisk.
Figure 4.11 Structured diagram of the different feedback given to suggested features in FreePBX.
Presentation of results

In the table below, all existing issues in the selected projects are divided into bugs or suggestion of features. The subsequent row is the percentage division between the two categories. The two last rows for each project is how many of the bugs and feature suggestions that have been initiated by the internal developers or core developers.

<table>
<thead>
<tr>
<th></th>
<th>Number of bugs</th>
<th>Suggested features</th>
<th>Total number or percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Qt</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issues</td>
<td>11771</td>
<td>3424</td>
<td>15195</td>
</tr>
<tr>
<td>Percent</td>
<td>77,5%</td>
<td>22,5%</td>
<td>100%</td>
</tr>
<tr>
<td>Internal</td>
<td>2907</td>
<td>923</td>
<td>3830</td>
</tr>
<tr>
<td>Internal, percent</td>
<td>19,1%</td>
<td>6,1%</td>
<td>25,2%</td>
</tr>
<tr>
<td><strong>KDElibs</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issues</td>
<td>5230</td>
<td>1245</td>
<td>6475</td>
</tr>
<tr>
<td>Percent</td>
<td>80,8%</td>
<td>19,2%</td>
<td>100%</td>
</tr>
<tr>
<td>Core dev</td>
<td>150</td>
<td>20</td>
<td>170</td>
</tr>
<tr>
<td>Core dev, percent</td>
<td>2,3%</td>
<td>0,3%</td>
<td>2,6%</td>
</tr>
<tr>
<td><strong>Asterisk</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issues</td>
<td>2244</td>
<td>29</td>
<td>2273</td>
</tr>
<tr>
<td>Percent</td>
<td>98,7%</td>
<td>1,3%</td>
<td>100%</td>
</tr>
<tr>
<td>Internal</td>
<td>192</td>
<td>5</td>
<td>198</td>
</tr>
<tr>
<td>Internal, percent</td>
<td>11,7%</td>
<td>5,8%</td>
<td>17,5%</td>
</tr>
<tr>
<td><strong>FreePBX</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Issues</td>
<td>3289</td>
<td>1367</td>
<td>4656</td>
</tr>
<tr>
<td>Percent</td>
<td>70.7%</td>
<td>29.4%</td>
<td>100.1%</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>Core dev</td>
<td>648</td>
<td>262</td>
<td>910</td>
</tr>
<tr>
<td>Core dev, percent</td>
<td>13.9%</td>
<td>5.6%</td>
<td>19.5%</td>
</tr>
</tbody>
</table>
Chapter 5

Evaluation and Discussion

In this chapter the results presented in chapter 4 will be discussed in order to answer the research questions stated in chapter 3 of this thesis. In addition to the discussion, this chapter also presents the limitations, validity and reliability in the conducted research.

5.1 Answering the Research Questions

The four projects have been reviewed through the three research questions in an equally manner. In this chapter, the results will be compared for the paired projects (hybrid + open source), the hybrid projects together and the open source together. Some discussions on which findings that are common for all projects are also given. For clarity, the research questions are presented in chronological order, as in the previous chapters.

5.1.1 Answers RQ1

\textit{H1.1. In all open source projects, most of the development activity is performed by a small group of core developers. In hybrid projects, these are often employees, in open source projects these individuals are often sponsored to work full-time with the project.}

\textbf{Qt and KDElibs}

Out of the 380 contributors to Qt, all contributors with more than two commits, a total of 191 persons were further examined. 114 of them could be identified as Qt/Nokia employees. From the remaining contributors, six were associated with two larger software consultancy companies. It is therefore reasonable to believe that the large majority of Qt contributors are paid for their contributions.

For KDElibs with its 580 contributors, the top 120 contributors, a little more than 20\% were selected as core developers, based on the 80/20 rule mentioned by Mockus et.al. These contributors stand for over more than 32.000 of the 36.000 commits, or about 89\% of all the commits.

One thing both of the projects have in common is somewhat the same number of contributors with over 500 contributions to the project, making both among the largest open source projects around. In order to have a rapid release schedule for the project, a critical thing in most open source projects, the contributors with the most contributions are to be regarded as key personal, and cannot just leave the project for a longer period of time. It should therefore come as no surprise that almost all contributors in these categories are hired or employed contributors. For Qt, 9 of the 10 top contributors were employed at Nokia, for KDElibs the number of sponsored key personnel was 2 out of 10. For the rest of the contributors, no information was found regarding workplace, so there is a change that the numbers are higher.
Asterisk and FreePBX

Asterisk and FreePBX are both smaller projects than Qt and KDElibs, and it were also harder to find any information about the contributors. Of Asterisks 86 contributors, 46 had more than two commits, and were closer examined; of these 17 were internal employees of Digium, while 29 are believed to be externals. Of the external contributors, two persons were working for partner companies of Digium Inc.

H 1.2 Most of the contributors that are non-core-developers or not employed in the hybrid software organization are users of other systems who rely on the hybrid open source/open source projects examined here. When they find bugs or missing functionality as a part of their normal use and operation of the project, they submit the bug(s), and/or request changes.

Qt and KDElibs

A sample of 20 persons were chosen randomly from the list of contributors, but in order to find information about the persons, the search were limited to contributors who have their own profile on ohloh.net. When selecting people who were among the top contributors were avoided. For Qt and KDElibs 20 out of 20 were involved in other OSS projects. For KDElibs all of these were related to KDE libs in some way. For Qt, half of the contributors’ projects were related to Qt.

The results here indicates that many of the contributors who are not active on a top level of the contributions, still are active in a variety of other projects. Number of side-projects per person varied from one and up to 50 different projects.

Qt and KDElibs

We can also see a division between the numbers of paid contributors in the hybrid project (Qt) versus the open source project (KDElibs).

Asterisk and FreePBX

Free PBX has a small number of contributors, but does also have many contributions between 3-49 commits, but in this case it is more likely to believe that the contributions come from volunteers.

A number of 10 contributors were randomly investigated from Asterisk, and out of them, five were involved in other projects. One of them was involved in Free PBX, but apart from that none of the projects were related.

10 contributors were also chosen from FreePBX 10 contributors were chosen. Here, four out of 10 were involved in other projects, and one of these was related to Free PBX.
5.1.2 Answers to RQ2

\[ H 2.1 \] Open source projects practices a wider degree of democracy when making decisions.

\[ H 2.2 \] Suggestions from internal contributors are seemed as more important than suggestions from external in hybrid projects. In open source projects, suggestions are more greeted and is more likely to be accepted and implemented

\textbf{Qt and KDElibs}

Qt has 1657 resolved issues on feature requests in its bug tracking system, the 1000 most recent of these were coded and the comments investigated.

The issues were mostly smaller changes and improvements to existing functionality and code base. The most interesting observation is that sometimes core developers both finds and fix the bugs. If one internal developer finds a bug and another internal developer corrects it, there are often no comments at all, indicating internal communication between the developers. In a few interesting cases, features reported were known to the persons correcting them, but no explanation were given on why this had not been corrected. Some of them were based on misunderstandings from the persons committing them, e.g. things that could be resolved using other methods or other APIs. However, some issues were closed without any comments at all, and most of them had one line with an explanation for the closure. In some cases the reporters were advised to use sub-optimal solutions to resolve their issues. In one case the person assigned to a task wrote that the issue was in fact a real issue, but it was not something Qt would focus on at the moment. The reporter of the issue refused the rejection, claiming that the issue could still be open but have low priority to be fixed in some future release. This was again rejected by another core developer stating that “Actually, we'd rather be honest and say that we are not working on something then keep bugs open with a lower priority when we really have no plans or intention of addressing them.” before closing the bug.

KDElibs uses Bugzilla to handle bug-reports and have its own section called wishlist. There are 614 closed entries on the wishlist, and 50 of them were closer investigated. 34 of the wishes were fixed, while eight of them were closed but not completely fixed. Four issues had workarounds that were believed to be good enough and another four were issues in other systems that KDElibs relies on. One interesting observation on KDElibs is the large number of comments every issue receives. Of the sample investigated one of the issues had 74 comments, while the average numbers of comments were between 5-10 for every issue. The high number of corrected bugs does probably correspond with the fact that a large number of people add fixes or new code to implement the feature wished for. Bugs are fixed in an unstructured manner and are not assigned to a specific core developer. This leads to the fact that many issues are resolved, and then the person who resolves it forget to change the status of the issue to fixed until later, or someone else in doing it later. Sometimes similar issues are reported and fixed, and when they are in place the similar bug is marked as corrected. Sometimes issues stay open for such a long time that they become obsolete because new versions of the framework appear. The most extreme case was an issue created in late 2001, commented for the first time in 2005, and then marked as solved in February 2010 because similar issues was solved in the meantime.
Asterisk and FreePBX

FreePBX uses Trac versioning system for its issues, and the bug database is organized in such a way that feature requests is its own category. This database contains 1048 feature request, which all were coded and analyzed.

Much of the same applies for Asterisk as for Qt. Many requests are cancelled or are welcome only if the developer can commit a working patch for the suggestion. Also many issues stays unresolved in the system for so long that the features end up as obsolete or unintended covered in a newer version of the software.

FreePBX have a remarkable high share of requests that are resolved without much interaction, and this is perhaps due to the fact that there are so few contributors that they might have other channels for communication and don’t really have the need to use the tracking system as a discussion board.

5.2 Limitations, validity and reliability

There are many sources of errors in the material. People, especially people contributing to open source projects, tend to use nicknames instead of their real names when communicating over the Internet. This might have caused people that are internal contributors to be counted as external in the hybrid projects.

Also a commit is not an accurate unit of measurement. It might be everything from changing one line of trivial code to adding a package with several lines with core functionality. Still for projects with many commits this has lesser relevance.

Surprisingly many of the contributors were only involved in just one project. I would have thought that at least the contributors in open source projects were involved in other projects. This result might be wrong and caused by the use of different nicknames for different projects, or it could be that people like to focus their effort on just one project.

The projects where most of the contributors were involved in other projects were KDElibs and Qt. One possible solution is that these projects are popular and attracts persons who have more time on their hands and perhaps has a higher technical knowledge. In cases where a person are involved in similar projects there is a possibility that errors discovered when integrating systems or lack of features in one of the similar projects is one of the reasons for joining a similar projects.
Chapter 6

Conclusions and Further Work

This chapter enlists the contribution and further work within this field of subject.

6.1 Contributions

Findings RQ1, summarized:
- Most top contributors are employed or hired to work with the project
- Hybrid projects have more employed/hired people than open source projects
- Depending on the size of the project, the contributors are involved in other projects.

Findings from RQ2, summarized:
- There are more comments in open source projects than in hybrid projects
- The overall friendliness and greetings of suggestions seems better in open source projects
- In most cases the reporter of an issue and the resolver are two different persons.
- The time-span for an issue to be fixed can be everything from minutes to several years, but are in most cases long
- In open source projects, tasks are rarely assigned to one person

For all the four projects examined, we can state that the top contributors always are paid for his or hers work, and this most likely also includes the core developers in open source project. Given the size of the projects examined, and the number of other contributors involved it is probably a great advance that a project, whether it is hybrid or open source, has one or more persons on the top who is updated on the codebase and bug tracking systems. On the other hand it is also a great advantage to have a very large number of more or less devoted volunteers to fix smaller bugs, or specific functionality that might be crucial just for a few users.

It is also clear that a person with many commits in a hybrid project is more likely to be employed within the host organization for the project, than a person with many commits in an open source project. There are large variances among hybrid and OSS projects, where open source projects have more commits from external contributors.

The overall impression is that features that are suggested in an open source project are much more welcome than similar suggestions in hybrid projects. This is clear from the number of suggestions that are accepted in open source projects and from the numbers of comments these receive compared with suggestions posted in hybrid projects. On the other hand, turning down a suggestion may not be an entirely evil act, but could also help keeping down the response time for more critical tasks to be resolved and to minimize the repositories.
The impression is that suggestions often are followed with a suggested solution. This applies both for open source and hybrid projects.

The dialogue and tone between the contributors also feels better in the open source projects than the hybrid projects. Nevertheless the language are polite and cheerful in all four projects.

6.2 Further work

As mentioned previously, hybrid projects are a relatively new way to organize software development projects on, and there is no indication that this will slow down in the future with continuously growing development costs and the industry’s fear of creating monster projects that is delayed for years before being cancelled at high cost.

An interesting project could be to do a study of only hybrid projects and look more into the differences and similarities between them. Another perspective could be to compare hybrid projects with an average closed, in house software project instead of open source project.

Finally, a discussion on open innovation and how it is linked with hybrid projects could also be an interesting study.
# Glossary

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Community</td>
<td>A group of contributors involved with the same (often Open Source) project</td>
</tr>
<tr>
<td>Free/libre Open Source Software</td>
<td>Software licenses in compliance with the “Free Software Definition” by the Free Software Foundation</td>
</tr>
<tr>
<td>Hybrid and Hybrid Open Source Software</td>
<td>An open source project managed by a commercial software company</td>
</tr>
<tr>
<td>Motivation</td>
<td>The psychological feature that arouses an organism to action toward a desired goal; the reason for the action; that which gives purpose and direction to behaviour</td>
</tr>
<tr>
<td>Open Source Software</td>
<td>Software that matches the requirements in Open Software Initiatives list “The Open Source Definition”</td>
</tr>
</tbody>
</table>
References

8 Maslow, Abraham H. (1943) A Theory of Motivation, Psychological Review, 1943
12 Conradi, Reidar, Li Jingye, Slyngstad, Odd Petter N, Kampenes, Vidgis By, Sjøberg, Dag, Morisio, Maurizio, Torchiano, Macro and Bunse, Christian, 2004 Survey on Risk Management in OTS-based Development in IT Industries in Several Countries p.3
15 Lerner, Josh and Tirole, Jean The Scope of Open Source Licensing, The journal of Law, Economics & Organization Volume 21, Issue 1, p.20-56. 2005
18 Ghosh, Rishab A. and David, Paul A. The nature and composition of the Linux kernel developer community: a dynamic analysis, 2003
19 Krishnamurthy, Sandeep Cave or Community?: An Empirical Examination of 100 Mature Open Source projects, University of Washington Bothell 2002
23 Bonaccorsi, Andrea and Rossi, Christina. Altruistic individuals, selfish firms? The structure of motivation in Open Source software. Sant Anna School of Advanced Studies Institute for Informatics and Telematics, 2002