

Study of the Release Process of Open Source Software

Case Study

Tor Erik Eide

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Reidar Conradi, IDI
Co-supervisor: Carl-Fredrik Sørensen, IDI

Problem Description

The student should study processes related to commercial release of open source software and identify criteria that are important for achieving a successful project.

Assignment given: 19. January 2007
Supervisor: Reidar Conradi, IDI

Abstract

This report presents the results of a case study focusing on the release process of open source projects initiated with commercial motives. The purpose of the study is to gain an increased understanding of the release process, how a community can be attracted to the project, and how the interaction with the community evolves in commercial open source initiatives.

Data has been gathered from four distinct sources to form the basis of this thesis. A thorough review of the open source literature has been performed. To further substantiate the data gathered from the literature study and to gain qualitative insights from companies heavily involved with open source development, four Norwegian companies adopting open source strategies have been interviewed. Data has also been gathered from active participation in the release process of the Keywatch networking software, including the creation of a web site and promotion of the project to build a community. Finally, the web sites of six company-initiated open source projects have been studied to gain further insight into how commercial open source projects are presented.

The contributions of this report can be divided into two parts; a description of the open source phenomenon and theoretical guidelines describing important measures to be taken into consideration when releasing software as open source. The description of the open source phenomenon is derived from reviewing the open source literature and includes a description of the history of open source, its characteristics, licenses, legal issues related to open source, and motivations for adopting open source software. The theoretical guidelines are based on corroboration of data gathered from qualitative interviews, reviewing of commercial open source web sites, and findings in the research literature. The guidelines are summarized in the concluding section of the report together with suggestions for future research.

Keywords: Open Source, Qualitative Research, Commercial Open Source Adoption, Open Source Preparations, Open Source Release Process, Open Source Community Management

Preface

This report is a Master Thesis at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU). The project presented herein is performed in the context of the ITEA COSI project (Co-development using inner and Open source in Software Intensive products).

I would like to thank my supervisors Professor Reidar Conradi and Dr. Carl-Fredrik Sørensen for their help and guidance for my work, as well as my co-advisor Audun Jensvoll from Keymind Computing. I would also like to thank Knut Yrvin from Trolltech, Stefan Landro and Christian Schwartz from Bekk Consulting, Thomas Malt from Linpro, and Vidar Langseid from eZ Systems for their willingness to participate in this study, and their valuable input through the interviews.

Finally, I would like to thank Øyvind Hauge for helpful feedback, and Per Kristian Schanke, my co-author in the depth project.

Trondheim, 2007-06-22

Tor Erik Eide

Contents

Abstract	i
Preface	iii
Contents	v
List of Figures	xi
List of Tables	xiii
I Research Context	1
1 Introduction	3
1.1 Background	3
1.2 Motivation and Rationale	3
1.3 Problem Description	4
1.4 Research Approach	4
1.5 Research Questions	4
1.6 Contributions	5
1.7 Report Outline	5
2 Research Design	7
2.1 Research Goal	7
2.2 Research Questions	7
2.3 Research Approach	8
2.3.1 Grounded Theory	8
2.3.2 Explorative Research	8
2.3.3 Descriptive Research	9
2.3.4 Sampling	10
2.4 Data Collection	11
2.4.1 Literature Survey	11
2.4.2 Participant Observation	13
2.4.3 Interviews	13
2.4.4 Transcription	14
2.4.5 Project Web Sites	14
2.5 Data Analysis	14

II	Pre-study	15
3	Open Source History	17
3.1	Research Laboratories	17
3.1.1	ARPANET and Network Infrastructure	17
3.1.2	Unix	18
3.2	GNU and the Free Software Foundation	18
3.3	Linux	19
3.4	World Wide Web	20
3.5	The Apache Software Foundation	20
3.6	The Open Source Initiative	21
3.7	The Mozilla Foundation	21
3.8	The Eclipse Foundation	22
3.9	The Impact of Open Source Today	22
3.10	Summary	23
4	Open Source Characteristics	25
4.1	Definition	25
4.2	The Community	26
4.2.1	Individual Motivations	26
4.2.2	Organizational Structure and Roles	28
4.3	The Development Processes	29
4.3.1	Conceptualization	29
4.3.2	Requirement Specification, Planning, Analysis and Design	30
4.3.3	Implementation	30
4.3.4	Writing Code and Submitting it to the Community for Review	30
4.3.5	Pre-Commit Test	30
4.3.6	Development Release	31
4.3.7	Production release	31
4.4	Summary	31
5	Licenses and Legal Issues	33
5.1	Intellectual Property	33
5.1.1	Patents	33
5.1.2	Trademarks	34
5.1.3	Copyright	34
5.1.4	Software Licenses	34
5.2	Open Source Licenses	35
5.2.1	Terminology and Common Properties	35
5.2.2	GNU General Public License (GPL v.2)	36
5.2.3	GNU Lesser General Public License v.2.1(LGPL)	37
5.2.4	Berkeley Software Distribution License(BSD)	37
5.2.5	Apache License	38
5.2.6	Mozilla Public License(MPL)	38
5.2.7	Eclipse Public License(EPL)	38
5.3	Summary	38

6	Commercial Adoption of Open Source	41
6.1	Commercial Use of Open Source Software	41
6.1.1	End-Use of Open Source Products	41
6.1.2	Reuse of Existing Components	42
6.2	Commercial Participation in Open Source Projects	42
6.2.1	Collaboration in External Projects	43
6.2.2	Releasing Code as Open Source	43
6.2.3	Development Assistance	43
6.2.4	Increased Adoption	44
6.3	Earning Money from Open Source	44
6.3.1	Support Sellers	44
6.3.2	Loss Leader	45
6.3.3	Widget Frosting	45
6.3.4	Accessorizing	45
6.3.5	Hybrid Model	45
6.3.6	Leveraging The Community	46
6.4	Summary	46
III	The Release Process of Open Source Projects	47
7	Identifying the Rationale	49
7.1	Motivations	49
7.1.1	Increasing the Value of the Product	49
7.1.2	Creating Revenues from Open Source	51
7.2	Open Source is Not a Magical Bullet	52
7.3	Summary	53
8	Important Measures to Achieve Success	55
8.1	Project Properties	55
8.1.1	The Project Name	55
8.1.2	Code that Works	56
8.1.3	Meet a Demand	57
8.1.4	Potential for Success	59
8.1.5	Software Architecture	59
8.2	Software Documentation	62
8.2.1	User Documentation	62
8.2.2	Development Documentation	63
8.3	Choice of License	63
8.3.1	Competitors and Complementary solutions	64
8.4	Summary	65
9	Releasing the Project	67
9.1	Tools Enabling Distributed Collaboration	67
9.1.1	Mailing Lists	67
9.1.2	Forums	68
9.1.3	Internet Relay Chat	68

9.1.4	Instant Messengers(IM)	68
9.1.5	Revision Control Systems	69
9.1.6	Bug Tracking Systems	69
9.1.7	Wikis	70
9.1.8	Choice of Tools	72
9.2	The Web Site	74
9.2.1	Layout and Design	75
9.2.2	Search Engine Optimalization	77
9.2.3	Web Site Content	78
9.3	Hosting the Web Site	81
9.4	Promoting of the Project	82
9.4.1	Developer Gatherings	83
9.4.2	Announcing on related Web Sites	84
9.4.3	Articles in the Media	86
9.4.4	Wikipedia	87
9.4.5	Actively engaging potential users and customers	87
9.4.6	Partner Sites	88
9.4.7	Newsletters	88
9.4.8	Paid Advertisements	88
9.4.9	External Effects	89
9.5	Summary	89
10	Community Management	91
10.1	Communication- and Development Activity	91
10.1.1	Communication	91
10.1.2	Web Site	93
10.1.3	Development Activities	93
10.2	Leadership and Organization	94
10.2.1	Commit Access	95
10.2.2	Symbiotic Relationship and Avoiding Conflicts	95
10.3	Providing Credits	96
10.4	Contributions from community	97
10.5	Summary	98
IV	Discussion	99
11	Evaluation of the Project	101
11.1	Evaluation of Research Questions	101
11.1.1	Research Question 1	101
11.1.2	Research Question 2	103
11.1.3	Research Question 3	105
11.2	Limitations and Validity Threats	106
11.2.1	Limitations	106
11.2.2	Threats to Conclusion Validity	107
11.2.3	Threats to Internal Validity	109
11.2.4	Threats to Construct Validity	109

11.2.5 Threats to External Validity	109
11.3 Summary	110
12 Conclusions and Future Work	111
12.1 Conclusions	111
12.1.1 Literature Study	111
12.1.2 General Guidelines	112
12.1.3 Preparations for the Release	112
12.1.4 After the project has been released	112
12.2 Suggestions for Future Work	113
12.2.1 Quantitative Research	113
12.2.2 Case Study	114
V Appendices	115
A Interview Guide	117
A.1 Hensikt med intervjuet	117
A.2 Intervjuets kontekst	117
A.3 Gjennomføring og behandling av informasjon	117
A.4 Annet	118
A.5 Intervju-spørsmål	118
A.5.1 Introduksjons-spørsmål	118
A.5.2 Tiltrekke community	118
A.5.3 Prosesser rundt slipp av open source	119
A.5.4 Interaksjon med community	120
B Interviewees	121
B.1 Bekk Consulting	121
B.2 eZ Systems	121
B.3 Linpro	122
B.4 Trolltech	122
C Project Web Sites	123
C.1 Assessment Questions	123
C.2 Openbravo	123
C.3 TYPOLight	123
C.4 netWiser	123
C.5 OpenEMM	124
C.6 Visual WebGUI	124
C.7 Webload	124
D Keywatch Web Site	125
E Glossary	129

List of Figures

4.1	Roles in open source projects	29
9.1	Using SVN with Keywatch	70
9.2	The issue states in Bugzilla	71
9.3	Communication in the Keywatch mailing list, May 2007	74
9.4	Menu structure of the Keywatch web site	76
9.5	Hits from announcing Keywatch at Freshmeat	85
D.1	Keywatch web site in Lynx	126
D.2	Layout and design of the Keywatch web site	127

List of Tables

2.1	Process of Building Theory from Case Study Research	9
2.2	Publication Databases	12
5.1	Properties of some of the most used open source licenses.	39
9.1	Tools available at surveyed web sites	73
9.2	Development status at studied web sites	79
9.3	Documentation found at surveyed web sites	80
9.4	Use of SourceForge by studied projects	81
9.5	Hits from posting Keywatch at GWT DnD web site	84
9.6	Hits the first week from announcing Keywatch at Freshmeat	85
11.1	The purpose of tools used in open source development	104
11.2	Promotional activities for open source projects	105
11.3	Commercial accommodation and adaption to the open source community	107
12.1	Guidelines related to general knowledge of open source	112
12.2	Guidelines related to preparations for the release of open source software	113
12.3	Guidelines related to the period after the release of open source software	114

Part I

Research Context

Chapter 1

Introduction

1.1 Background

This project is performed in the context of the ITEA-COSI Project. The COSI project is an international project driven by the industry. The COSI project has observed a shift in the development process of complex software, including a shift toward commercial collaboration with open source development communities¹. As a participant in the ITEA-COSI project, NTNU contributes with research to better understand the industry's use of open source software.

This project is designed to provide a better understanding of the release process of commercial open source initiatives. It consists of a literature review, the finding from four interviews with Norwegian open source adopters, and data gathered from the project portals of six commercial open source initiatives. This project extends the work of [ES06], that identified a set of guidelines for companies considering to release their software as open source. Based on new findings, these guidelines are further refined and evaluated.

This thesis concludes the MSc in Computer Science at the Department of Computer and Information Science (IDI) at the Norwegian University of Technology and Science(NTNU).

1.2 Motivation and Rationale

Open source has received a large amount of attention in the last several years, being characterized as a fundamentally new way to develop software [Ray02, MFH02]. New open source projects are released every day, and the amount of open source adopters is steadily growing². Since the coining of the open source term in 1998, open source has gained an increasing economic importance. An increasing number of companies have released code created under proprietary conditions in the hope of growing a community to improve and maintain the future code base [WM05].

The amount of research literature on open source software is immense and continually growing.

¹<http://www.itea-cosi.org>

²Shown ,e.g, by the amount projects and users at sites such as <http://www.sourceforge.net>, Registered Projects 2006-12-18: 137,080[ES06], Registered Projects 2007-06-04: 150,109

There is, however, a general lack of research on commercially sponsored open source projects [WM05]. As open source software becomes a development model in conjunction with traditional software development, more attention needs to be paid to this area of research. There is not much literature related to how commercial actors initiate new open source projects. Some literature on commercial open source adoption can be found, but it has mostly focused on sponsoring of already well-established projects or those initiated by large actors well known for their existing proprietary solutions with an established customer base [SFT07], e.g., [MFH02] [LT02] [Wes03] [Sam06]. Additionally, there is a need for more qualitative research on open source processes [Sca07]. While quantitative research methods stress collection and analysis of data that is usually easy to quantify, not all phenomena related to open source are readily captured or characterized in quantitative form [Sca07].

Thus, this study concentrates on newly started projects initiated by commercial actors, and the challenges these face in the initial phase of going open source, taking a qualitative research approach. Some of the major goals of all open source projects are to increase community size, improve internal collaboration and to further develop the software [Stü06]. Considering the amount of new open source projects that are initiated every day, it can be a difficult challenge for open source initiators to distinguish their open source project from those that already exist. This challenge induces a need for more research on how to attract and build a community. Being particularly interested in the strategies and processes by which open source projects initiated by commercial actors attract a community, issues such as the preparations for the release, the initial presentation of the project, marketing of the project and how to build and sustain a community is presented in the thesis.

1.3 Problem Description

The student should study processes related to commercial release of open source software and identify criteria that are important for achieving a successful project.

1.4 Research Approach

It was decided to adopt a case study approach studying the processes related to commercial release of open source qualitatively. The case study was conducted by gathering evidence from four separate sources, with totally 11 cases. This included a literature survey, interviewing four separate companies, active participation in the release process of an open source project, and examining the web sites of six commercial open source initiatives.

1.5 Research Questions

The following research questions were defined according to the problem description:

- RQ1: What considerations and preparations are important to make in advance when releasing an open source project?

-
- *RQ1.1* What is the rationale behind commercial release of open source software?
 - *RQ1.2* What measures are important to consider before a project is released?
 - RQ2: What impact will the processes applied when releasing software as open source have on the success of the project?
 - *RQ2.1* What comprises a good technical infrastructure, and how important is the provision of a good infrastructure to attract a community?
 - *RQ2.2* What is the importance of providing a good website for an open source project?
 - *RQ2.3* What specific promotional activities are performed by commercial open source actors, and what is the effect of such activities?
 - RQ3: How do commercial actors manage and interact with the open source community?
 - *RQ3.1* How important is the interaction and project activity to attract and sustain a community?
 - *RQ3.2* How do companies accommodate and adapt to the community in order to attract and sustain it?

1.6 Contributions

The contributions of this report can be divided into two parts; a description of the open source phenomenon and theoretical guidelines describing important measures to be taken into consideration when releasing software as open source. The description of the open source phenomenon is derived from reviewing the open source literature and includes a description of the history of open source, its characteristics, licenses, legal issues related to open source, and motivations for adopting open source software. The theoretical guidelines are based on corroboration of data gathered from qualitative interviews, reviewing of commercial open source web sites, and findings in the research literature. The guidelines are summarized in the concluding section of the report together with suggestions for future research.

1.7 Report Outline

The report is divided in five parts, comprising twelve chapters and X appendices.

Part I contains the context of the project, and comprises Chapter 1 and 2. Chapter 1 contains the introduction to the project, outlining the background, the motivation and rationale, the problem description, the contributions, and how the report is presented. Chapter II contains a description of the Research Design, including the goal of the project, the research questions and their rationale, the research approach, the data collection procedure, and how these data will be analyzed.

Part II contains the pre-study of the project, and comprises Chapter 3, 4, 5, and 6. Chapter 3 presents the evolution of the history of open source, from its roots in the research laboratories in the late 60s to its present state. Chapter 4 provides a description of the characteristics of open

source. These characteristics include the definition of open source, a description of open source communities, and the development processes found in open source projects. Chapter 5 explains licenses and legal issues in relation with open source. It presents the concept of intellectual property, and provides a description of common properties found in open source projects. It also contains a review of six open source licenses that are commonly used. Chapter 6 presents issues related to commercial adoption of open source software. It describes how open source can be used, and how participation in open source projects can provide benefits to a commercial actor. This chapter also includes a description of business models that can be adopted by companies in relation with open source.

Part III contains a description of the release process of open source software, and comprises chapter 7, 8, 9, and 10. Chapter 7 presents the importance of identifying the rationale, describing motivations found from the interviews, and the importance of being aware of the work associated with initiating an open source project. Chapter 8 presents measures that are important to consider to achieve a successful project. It describes project properties that should be present, the importance of providing proper documentation, and issues related to the choice of an open source license. Chapter 9 presents the issues that should be taken care of in relation with the release of an open source project. It describes the tools that can enable distributed collaboration, the importance of providing a good web site, and promotional tactics that can be used to create awareness and interest of an open source project.

Part IV contains the discussion of the project, and comprises Chapter 11 and 12. Chapter 11 presents an evaluation of the project, including an evaluation of the research questions, and an evaluation of the limitations and validity threats identified in the project. Chapter 12 presents the conclusions and suggestions for future work.

Part V contains the appendices, and comprises Appendix A, B, C, D, and E. Appendix A presents the interview guide, including an introduction letter and the questions that were used as an aid when performing the interviews. Appendix B presents the interviewees and provides a brief description of the companies, their open source project(s), and the interviewees' roles in the companies. Appendix C presents the assessment questions, and the projects that were presented at examined web sites. Appendix D presents how the Keywatch web site was presented in Lynx, and the initial layout and design. Appendix E contains the glossary where abbreviations used in this report are presented.

The Bibliography is presented as the final component of this thesis.

Chapter 2

Research Design

This project intends to provide further insights into the initiation of open source projects from small commercial actors. To accomplish this, a case study approach has been chosen. This chapter describes the research questions, the research approach, the data collection approach and gives a description on how the collected data has been analyzed.

2.1 Research Goal

The goal of this research is to describe the processes of commercial open source initiatives with a focus on the early stages of the release process. This includes identifying the rationale behind commercial open source initiatives, and what companies can do to increase their chance of having a successful project.

2.2 Research Questions

Early definition of research question(s) is important when building theory from case studies [Eis89]. The research questions were created in accordance with the problem description and goal of the project, each question pertaining to specific subject of interest to achieve a successful open source project. Three research questions were defined, and subquestions were created to elaborate specific areas of interest.

- RQ1: What considerations and preparations are important to make in advance when releasing an open source project?
 - *RQ1.1* What is the rationale behind commercial release of open source software?
 - *RQ1.2* What measures are important to consider before a project is released?
- RQ2: What impact will the processes applied when releasing software as open source have on the success of the project?

- *RQ2.1* What comprises a good technical infrastructure, and how important is the provision of a good infrastructure to attract a community?
 - *RQ2.2* What is the importance of providing a good website for an open source project?
 - *RQ2.3* What specific promotional activities are performed by commercial open source actors, and what is the effect of such activities?
- RQ3: How do commercial actors manage and interact with the open source community?
 - *RQ3.1* How important is the interaction and project activity to attract and sustain a community?
 - *RQ3.2* How do companies accommodate and adapt to the community in order to attract and sustain it?

2.3 Research Approach

Case study is a research strategy that focuses on understanding the dynamics present within single settings. A case study can involve either single or multiple cases [Eis89], where a variety of different data collection procedures may be applied [WRH⁺00]. Multiple cases are a powerful means to create theory because they permit replication and extension among individual cases [Eis91], and thus a multiple case approach was chosen. This included participational observation of an open source project, interviews of four commercial open source adopters, and evaluation of web pages from six open source projects.

2.3.1 Grounded Theory

The research process adopted is based on the concept of “Grounded Theory”, which refers to the creation of theory from inductive reasoning and comparative analysis. There are three basic elements of grounded theory; concepts, categories, and propositions. The concepts are formed into conceptual categories, then the evidence from which the category emerged, is used to illustrate the concept [GS67]. Grounded theory can be presented either as a well-codified set of propositions or in a running theoretical discussion [GS67]. This report presents the theory in a running theoretical discussion, presenting the emerging categories as theoretical guidelines. The general research approach that has been followed is outlined in Table 2.1.

2.3.2 Explorative Research

Explorative research can be used as a pre-study for a more through investigation of a topic to assure that important issues are not overseen [WRH⁺00]. In explorative research, neither the nature nor the dimensions of a topic are well known [KW01]. Thus, this project takes an explorative approach to get a better understanding of commercial open source practices and, and the release process of commercial open source software in particular.

Step	Activity	Reason
Getting Started	Definition of research question Possibly a priori constructs	Focuses efforts Provides better grounding of construct measures
Selecting Cases	Neither theory nor hypotheses Specified population	Retains theoretical flexibility Constrains extraneous variation and sharpens external validity
	Theoretical, not random, sampling	Focuses efforts on theoretically useful cases—i.e., those that replicate or extend theory by filling conceptual categories
Crafting Instruments and Protocols	Multiple data collection methods	Strengthens grounding of theory by triangulation of evidence
	Qualitative and quantitative data combined Multiple investigators	Synergistic view of evidence Fosters divergent perspectives and strengthens grounding
Entering the Field	Overlap data collection and analysis, including field notes	Speeds analyses and reveals helpful adjustments to data collection
	Flexible and opportunistic data collection methods	Allows investigators to take advantage of emergent themes and unique case features
Analyzing Data	Within-case analysis	Gains familiarity with data and preliminary theory generation
	Cross-case pattern search using divergent techniques	Forces investigators to look beyond initial impressions and see evidence thru multiple lenses
Shaping Hypotheses	Iterative tabulation of evidence for each construct	Sharpens construct definition, validity, and measurability
	Replication, not sampling, logic across cases	Confirms, extends, and sharpens theory
	Search evidence for "why" behind relationships	Builds internal validity
Enfolding Literature	Comparison with conflicting literature	Builds internal validity, raises theoretical level, and sharpens construct definitions
	Comparison with similar literature	Sharpens generalizability, improves construct definition, and raises theoretical level
Reaching Closure	Theoretical saturation when possible	Ends process when marginal improvement becomes small

Table 2.1: Process of Building Theory from Case Study Research [Eis89]

2.3.3 Descriptive Research

Descriptive research can be performed to provide a systematic description that is as correct as possible. Given the goal of this project, the release process of commercial open source projects and what can be done to increase the chance of having a successful project is described.

2.3.4 Sampling

The concept of population is crucial as it defines the set of entities from which the research sample is to be drawn [Eis89]. The population of interest in this study is commercial actors that have released open source projects. The selection of cases is an important procedure in case study research, and two sampling procedures were conducted; convenience sampling and theoretical sampling.

Convenience Sampling

Convenience sampling relates to selecting the most convenient subjects [WRH⁺00], and were conducted to select the interviewees. All the respondents were employees of Norwegian companies that have adopted an open source strategy, and were selected from the defined population through the supervisor's contacts.

The interview subjects were:¹

- Knut Yrvin (Community manager) - Trolltech
- Stefan Landro (Leader of the open source group) and Christian Schwartz (Open source project founder) - Bekk Consulting
- Vidar Langseid (Head of development) - eZ Systems
- Thomas Malt (Substitute and forthcoming head of development) - Linpro

Additionally, it was decided to partake in the release of the open source project described in [ES06], Keywatch. Being involved with the Keymind² and the project the last several months, this was a natural project of which to be a participating observer.

Theoretical Sampling

The goal of theoretical sampling is to choose cases which are likely to replicate or extend the emergent theory [Eis89], and this approach was taken in the selection of the commercial open source projects that were to be studied. The following criteria were established in selecting the projects:

- Freshmeat was selected as the medium for finding the projects³.
- 6 projects should be chosen⁴.
- The projects should be initiated by commercial actors.

¹A short description of the companies they represent, and the projects that were the main subject of their interviews can be found in Appendix B

²Keymind is the company that has created Keywatch.

³<http://freshmeat.net>

⁴In adherence with [Eis89], that described any number between 4 and 10 as a good number of cases

-
- The projects should be publicly released as open source in 2006 or later, to ensure that the projects were newly initiated⁵.
 - A canned hosting service, such as SourceForge, should not be used for the main presentational web page.
 - Archives from mailing lists and/or forums should be publicly available.
 - The projects should give the impression of being successful.

By going through release announcements at Freshmeat, several projects were found to fulfill these criteria. These projects were then studied to ensure that the website's contained enough information to analyze. 12 projects remained after this process, and six of these were chosen for the study⁶:

- netWiser
- Openbravo
- OpenEMM
- TYPOLight
- Webload
- Visual WebGUI

2.4 Data Collection

Data was collected from four different sources; the research literature, interviews, participant observation, and project websites. Using several sources of knowledge is a principle described by [Yin03]. When the pattern from one source of data is corroborated by the evidence from another, the findings are stronger and better grounded [Eis89], and the triangulation made possible by multiple data collection methods provides stronger substantiation of constructs, increasing construct validity [Eis89, Yin03].

2.4.1 Literature Survey

The literature survey builds on the pre-study performed in [ES06], in which a generalized literature survey on open source were performed. This study is extended with the inclusion of more recent studies, a targeted search on commercial open source, as well as the addition of new findings found in literature that was not surveyed the last time. The intention of the literature survey is to further substantiate the results gathered from the other data sources.

⁵The first year is a critical phase in establishing sufficient momentum for the project to mobilizing newcomers [vKSL03].

⁶Descriptions of the selected projects can be found in Appendix C.

Finding the Literature

Three major sources were used for finding the literature, Google Scholar, publication databases, and citations found in the literature.

Google Scholar

Google Scholar⁷ covers peer-reviewed papers, theses, books, abstracts, and other scholarly literature from a variety of academic publishers and professional societies, as well as scholarly articles available across the web⁸.

Google scholar was used mostly to perform a wide search in several publication databases without locking each search to a specific database. As open source encompasses several fields of research outside computer science, it was especially useful to find interesting publications in journals focusing on other areas than computer sciences, such as the economic and social sciences. It also provided access to some drafts of which has been published in journals, but were found inaccessible due to access limitations.

Publication Databases

Electronic journal databases have been a major source for this thesis, the most used databases being:

Journal Database	URL
The ACM Digital Library	http://portal.acm.org/portal.cfm
ScienceDirect	http://www.sciencedirect.com/
IEEE Xplore	http://ieeexplore.ieee.org/Xplore/dynhome.jsp
Blackwell Synergy	http://www.blackwell-synergy.com/

Table 2.2: Publication Databases

Citations

Citations from the literature were used to find other interesting publications. It is often difficult to indicate the relevance of open source literature based on their titles, and thus citations were important in finding literature that were not found through regular searches in the databases.

Other Sources

Google⁹ and Wikipedia¹⁰ were used in addition to the mentioned sources for gaining a general overview of certain topics. Being aware of the limitations of these sources due to lack of quality assurance and reviews, findings from these sources were not cited in the thesis. Thus, the sources of information found in these media were validated and cross-checked with the published literature or other reliable sources.

⁷<http://scholar.google.com>

⁸Paraphrased from <http://scholar.google.no/intl/en/scholar/help.html>

⁹<http://www.google.com>

¹⁰<http://www.wikipedia.com>

2.4.2 Participant Observation

Participating observation was performed in relation to the release of Keywatch as open source software. This included the creation of an initial design of their project web site. It also included providing advice and suggestions regarding the release process, announcing the project, and the appliance of certain marketing strategies.

By taking an active part in the entire process, with regular telephone meetings and email contact, an inside view on releasing open source has been obtained. It was initially intended that this process should involve several measurable steps in relation with the iterative process described in [ES06]. However, due to external factors, this has not been possible. Thus, the data gathered from this process, consist of the results of actions that were taken in relation with the announcement and marketing aspects, as well as the experiences in relation with this process. This has enabled the creation of an in-depth picture on releasing open source software.

2.4.3 Interviews

The interview process consisted of three steps; preparation, execution and transcription.

Interview Preparation

A general interview guide approach was chosen for conducting the interviews. The general interview guide approach is intended to ensure that the same general areas of information are collected from each interviewee; this provides more focus than the conversational approach, but still allows a degree of freedom and adaptability in getting the information from the interviewee [VS02]. The interviews were thus thematically guided toward the release of open source projects and open source adoption of commercial actors. The initial questions were created and reviewed by my supervisors. They were then modified in accordance with suggestions from the supervisors and went through another review process to reach the final questions.

When the interview guide was finished, and upon receipt of confirmation from the interviewees, the further preparations consisted of the following steps, as described in [VS02]:

- Choose a setting with the least distraction.
- Explain the purpose of the interview.
- Address the terms of confidentiality.
- Explain the format of the interview.
- Indicate how much time the interview will require.
- Provide contact information to the interviewees.
- Allow interviewees to clarify any doubts about the interview.
- Prepare a method for recording data.

It was decided to use a digital voice recorder to record data, and a preparation letter addressing the mentioned steps were sent to the interviewees¹¹. Providing an introduction letter that describes the interview process and intentions is an important step in conducting an interview, as this increases trust, lets the interviewees know how the interviews will be conducted, as well as provides an opportunity to answer questions before the interviews take place.

Interview Execution

The interviews were conducted in the course of one week, and each interview took 45-90 minutes. All the interviews were recorded with a digital tape recorder to facilitate for complete transcription. There were certain challenges in performing the interviews, such as lack of interview skill and problems with following the predefined questions. This was a result partly due to lack of experience of the interviewer and partly because the interview guide approach accommodates for informal talk. However, the interviews were a successful method for gaining important insights, and provided a valuable contribution to the project.

2.4.4 Transcription

The interviews were completely transcribed to facilitate data analysis. This process was highly iterative as the recordings had to be played through several times to be able to transcribe the entire interaction. The data was then analyzed and interesting quotations were translated into English for use in this report.

2.4.5 Project Web Sites

Assessment questions¹² were created to identify what commercial open source project sites present to their community, and the findings were codified in tables for use in the report.

2.5 Data Analysis

A comparative data analysis strategy has been chosen to analyze the data. The data gathered has been searched for cross-case patterns to identify conceptual categories. The data from the qualitative interviews, from reviewing of commercial open source web sites, and from findings in the literature have then been corroborated to create theoretical guidelines.

¹¹The introduction letter can be found in Appendix A (In Norwegian)

¹²The studied projects and the assessment questions can found in C.

Part II

Pre-study

Chapter 3

Open Source History

Open source has evolved to its current state as a result of several technological innovations and breakthroughs in the domain of digital communication, as well as through new legal innovations leading to the emergence of a new software development model and new business models. To understand open source software, the open source licenses, business models, and motivations for adopting open source, it is important to have an understanding of its history. Thus, this chapter describes how the history has formed open source to its present state.

3.1 Research Laboratories

Open source can be traced back to the early days of computing, where most of the software development was performed in academical and corporate research laboratories [vHvK03, LT02]. At this time, software was typically embedded in highly expensive computers only available to research laboratories, and it was considered normal practice to freely distribute the software being developed. This made it possible for people to collaborate on the development efforts across organizational boundaries, building upon each other's research. The research environments were responsible for the creation of both the ARPANET and UNIX, which can be viewed as the foundation of a new way of thinking about computers. Computers were no longer devices only for logical computation, but rather platforms for communication and collaboration [Ben05]. Many of the organizations composing this community used incompatible hardware platforms, thereby leading to much focus on establishing platform-independent network technologies that could work across organizational borders.

3.1.1 ARPANET and Network Infrastructure

The ARPANet¹ was created in 1969, being the world's first packet-switched network. This network, which is now widely recognized as the forerunner to the Internet [MS02], linked together several major computer facilities such as universities, research laboratories, and defense contractors [Lin04]. The amount of interconnected institutions quickly increased, and during 1975 it reached more than

¹Advanced Research Projects Agency Network

100 institutions, making this a natural medium for communication.

Much of the communication focused on establishing platform-independent technologies, and this took place using the informal correspondence process known as Request For Comments (RFC). This quickly became the standard way of communicating ideas and for providing comments and refinements to existing proposals within the technical community of the ARPANet [MS02]². The RFCs proved vital to the evolution of the network infrastructure, and lead to the emergence of a huge amount of protocols that are still in use today. One of protocols that were refined in the RFCs³, was the Transmission Control Protocol (TCP)⁴. The TCP provided the means to connect physically distinct networks, while being highly reliable and featuring an open architecture. Hence, it proved to be very well suited for integrating networks built on a variety of platforms and protocols [MS02].

3.1.2 Unix

Resources were also directed toward the development of the platform-independent operating system; Unix [LT02]. Unix was initially created by Dennis Ritchie and Ken Thompson in 1969 at the AT&T's Bell laboratories, and was distributed freely to other organizations which helped developing the software further [LT02]. One of these organizations was the University of California (Berkeley), which established the Berkeley Software Distribution (BSD) of Unix in 1977. The Berkeley distribution became very popular, and was the operating system of choice to link together the ARPANet-nodes. When version 4.2 of the Berkeley Operating System was released in 1983, it included an implementation of the TCP protocol. This contributed heavily to the adoption of TCP as the communication protocol in the ARPANet [MS02], and the old protocol was replaced the same year. 1983 was also the year when Larry Wall created the patch utility, that made it possible to patch in contributions to development efforts without providing the entire source code, arguably being the most important single tool to enable collaborative distributed development across the ARPANet [Ben05].

3.2 GNU and the Free Software Foundation

During these early days, there were typically not spent any resources defining property rights or to restrict the reuse and distribution of software. This changed in the 1980s. The first modern PC was created by IBM in 1981, and the interoperability induced by the IBM-PC standard endorsed the creation of software that could run on different machines [Ben05]. This enabled the market for proprietary software, and competing corporations started creating software where the source code was kept secret. The trend had turned toward proprietary source code. An organization that adopted this trend was the AT&T's Bell laboratories which began enforcing their claimed intellectual property rights on Unix. This resulted in a court battle with BSD over copyright

²The RFCs have many commonalities with open source software, e.g., they evolve as a collaborative effort with peer review

³A huge amount of the protocols and the network structure has been refined in the RFCs, and a full listing can be found at <http://www.rfc-editor.org/rfc-index2.html>

⁴Initially created in 1973. was refined in the RFCs the following years

violation that was not resolved until the early 1990s [FF02]. At about the same time, MIT⁵ licensed some of the code developed by their hackers to a commercial firm, restricting access to the source code even for those who had participated developing it [vHvK03].

Richard M. Stallman, a programmer based at the MIT laboratories did not favor this trend of proprietary software⁶:

“The most obvious possibility was to adapt myself to the change in the world. To accept that things were different, and that I’d just have to give up those principles and start signing non-disclosure agreements for proprietary operating systems, and most likely writing proprietary software as well[.]”

Developing proprietary software would be misusing my skills[.] I decided to look for some other alternative. What can an operating system developer do that would actually improve the situation, make the world a better place? And I realized that an operating system developer was exactly what was needed. The problem, the dilemma, existed for me and for everyone else because all of the available operating systems for modern computers were proprietary. “

Stallman was led to the concept of free software, and he decided to create a complete operating system, with all necessary software tools - The GNU Project. Stallman resigned from MIT in January 1984, and began developing for GNU. The first tool, GNU Emacs, was released early 1985, and soon people started reporting bugs and offering fixes and new features to the software.

To protect GNU from being used in proprietary packages and to “*promote computer users’ rights to use, study, copy, modify, and redistribute computer programs*”⁷, Stallman founded the Free Software Foundation in 1985. He also created a new form of intellectual property manifested in the GNU General Public License (GPL). This license, being quite revolutionary at that time, used copyright law to ensure that all users could use the software at no cost while at the same time allowing everyone to freely modify and redistribute the software. To preclude commercialization of cooperatively developed software, GPL-ed code could not be included in proprietary software. This was made possible with the use of a concept commonly referred to as copyleft⁸.

3.3 Linux

During the next years, Richard Stallman and the Free Software Foundation managed to develop all of the components of GNU, except for the kernel⁹. In 1991, a student named Linus Torvalds began developing the kernel of what is today known as Linux. His objective was to create a UNIX-like operating system for the IBM-PC 386 series [FF02]. In this process, he openly sought help, and he succeeded in attracting a great deal of worldwide support. The GNU components were integrated

⁵Another university that had participated in source-code sharing

⁶From the transcripts of Richard M. Stallman’s speech, “Free Software: Freedom and Cooperation”, available at <http://www.gnu.org/events/rms-nyu-2001-transcript.html>

⁷From: <http://www.fsf.org>

⁸The GPL and copyleft is further explained in the next chapter.

⁹The core of the operating system that manages the basic operations

with the kernel in a large-scale collaborative effort. This was made possible with the emergence of the Internet and the World Wide Web. Indeed, the Linux development was characterized by almost completely relying on Internet-based tools [Tuo01]. Torvalds also introduced an entirely new development model, in that unstable versions of the project were released early and often. This has later been defined as a fundamental characteristic of success of the free software development process [Ray02].

3.4 World Wide Web

The proposal of the World Wide Web (WWW) was published by Tim Berners-Lee in 1991, and unveiled an entirely new technological scenario based on distributed communication and pervasive connectivity to a technological network; The Internet [Ben05]. The WWW was based on the existing model of the hypertext first published in 1965 [Nel65]. By connecting the concept of hypertext to the network architecture of the ARPAnet, an extremely important invention in the open source history had been created. The World Wide Web was conceived as an open architecture based on open standards, making it possible for everyone to create software for this new media without licensing restrictions. At the same time, the network backbone was continually evolving with significant speed enhancements, and the cost of personal computers were reduced, making them common in people's homes [MS02]. Together, these enhancements caused an exponential increase in the use of the ARPAnet, or what was now commonly referred to as the Internet¹⁰. The increased use of the Internet allowed for increased diffusion of software and an increase in the contributions to freely share software development projects, such as Linux [LT02].

3.5 The Apache Software Foundation

A highly successful project adopting distributed development with free sharing of the source code, is the Apache HTTP Server. The development of the Apache project started in February 1995, due to the discontinuation of the development efforts of the already existing NCSA¹¹ web server. The Apache development was based on a series of patches to the NCSA server, and was performed by a group of volunteers, known as the Apache Group. This group then worked together to coordinate the distribution of the patches.

In 1999, the Apache Group founded the Apache Software Foundation, and created the Apache Software License. The Apache Software Foundation is today the host of several other high-profile projects in addition to the web server¹². The Foundation was formed to:

- Provide a foundation for open, collaborative software development projects
- Create an independent legal entity
- Provide a means for individual volunteers to be sheltered from legal suits

¹⁰Statistics are available at <http://www.isc.org/index.pl?/ops/ds/>

¹¹National Center for Supercomputing Applications

¹²<http://www.apache.org>

To avoid lawsuit for breaches of copyrights and patents, each developer must give Apache unlimited licensing rights to the material provided to its projects.

3.6 The Open Source Initiative

While gaining media attention, and being adopted across the world, free software was not yet an industrial success. In 1998, Eric Raymond released the first draft of the highly influential book, “The Cathedral and the Bazaar” [Ray02], describing the nature of free sharing, and why it was a successful development model. After reading this text, Netscape as the first commercial actor, announced the release of the source code of their proprietary software, the Netscape Navigator Browser. Netscape had adopted the open source model to deny Microsoft a monopoly lock on the browser market, and indeed this goal has been achieved. Since Netscape released their software as open source, there have been a tremendous explosion of interest in the open source development model. It can be said that this move was a defining moment for the open source movement.

A group of people that met on a regular basis, consisting of among others Eric Raymond, was interested in spreading the awareness of the tools that had been developed outside the proprietary development model. When Netscape announced their release, these proponents decided to adopt a more commercialized stance than the ideologies of the Free Software Foundation. According to Bruce Perens, they “realized it was time to dump the confrontational attitude that has been associated with “free software” in the past and wanted to sell the idea strictly on the same pragmatic, business-case grounds that had motivated Netscape” [ES06]. Several people went together and coined the term “Open Source”. With this, they hoped to get the corporate world to listen to their claims regarding the superiority of an open development process. The Open Source Initiative (OSI), was launched the following week. They also created the Open Source Definition, enabled the concept of open source certification, and created a list of licenses that met the standard for such a certification based on the Debian Free Software Guidelines. After the founding of the OSI, things started happening in the commercial world.

When Netscape later released their source code, they issued a press release where the open source term was used. The same year, several other companies decided to approach open source. IBM was one of the first adopters of open source, stating they would sell and support Apache as a part of their WebSphere project in June 1998, while Oracle announced that they would port their database to Linux [Ben05]. Even Microsoft began taking notice of the open source movement, and the last week of October 1998, a confidential Microsoft memorandum on their strategy against Linux and Open Source software was leaked¹³.

3.7 The Mozilla Foundation

Mozilla was released as an open source version of Netscape in January 1998. Since then, Mozilla has released many version of its browsers, where Firefox is the latest [Kri05]. Together with the source code, they also supplied an open source license, the Mozilla Public License, a license that is used with all projects hosted by Mozilla. The Mozilla Firefox browser is today the second most used

¹³Known as the “Halloween Documents” <http://www.catb.org/~esr/halloween/index.html>

web-browser, only preceded by the Microsoft Internet Explorer¹⁴ [App07]. The Mozilla Foundation was established in July 2003 as a non-profit corporation dedicated to public benefit. The Mozilla Corporation was subsequently established in August 2005, as a subsidiary to the foundation to coordinate the development and marketing of Mozilla technologies and products [moz07b]. Today, the Mozilla Foundation is one of the largest distributors of open source software [moz07a].

3.8 The Eclipse Foundation

The Eclipse open source project was originally created by IBM in November 2001, being supported by several other software vendors; Borland, MERANT, QNX Software Systems, Rationale Software, Red Hat, SuSE, TogetherSoft, and Webgain. This consortium grew rapidly and reached more than 80 members by the end of 2003 [Fou07a]. To provide an independent non-profit foundation acting as a steward to the Eclipse community, the Eclipse Foundation was created in January 2004. The foundation was formed with the intention to provide an open, transparent community that was vendor neutral [Fou07a]. This foundation provides four services; Infrastructure, intellectual property management, development process, and an ecosystem for open source software[Fou07a]. All technology provided to and developed by the eclipse community is licensed under the Eclipse Public License.

3.9 The Impact of Open Source Today

In the years following the founding of the OSI, the impact of open source software has been continually increasing. It is now deeply affecting the industrial dynamics in the software industry [DL03]. Several commercial actors have followed in the footsteps of Netscape and decided to release their proprietary software as open source. Additionally, the adoption of open source software in commercial development efforts has been a well-known phenomenon [LT02][Fit06]. This is exemplified by the commercial interest in the Eclipse foundation, today having a very large base of members¹⁵, as well as the sponsoring of the Apache Foundation by companies such as Google, Hewlett Packard, and Covalent¹⁶. Additional examples include the large-scale adoption of middle-ware such as Spring¹⁷, Hibernate¹⁸, and the JBoss application server¹⁹. It should also be noted that Google has recently released the Google Web Toolkit as open source²⁰, and that the source code of the Java Development platform has been made available as open source, being a major contribution to the open source movement²¹. Open source is today often referred to as a development methodology in itself, that complements, and in some cases even competes with with traditional, commercial development practices [Hec00].

¹⁴The average person tends to use the Internet Explorer as it is provided alongside the Microsoft Windows Operating Systems

¹⁵<http://www.eclipse.org/membership/>

¹⁶<http://www.apache.org/foundation/thanks.html>

¹⁷<http://www.springframework.org/>

¹⁸<http://www.hibernate.org/>

¹⁹<http://labs.jboss.com/jbossas/>

²⁰http://googlewebtoolkit.blogspot.com/2006/12/gwt-13-release-candidate-is-100-open_12.html

²¹<http://www.sun.com/smi/Press/sunflash/2006-11/sunflash.20061113.1.xml>

Open source software is also gaining terrain in the “end-use” arena, both by individual persons and in many organizations. National governments are beginning to recognize the value of open source software, and are worldwide adopting open source solutions across the public sector. This is much due to the recent focus on open standards²², which are generally accepted as being better supported by open source software than their proprietary alternatives [Sch01]. Other factors, such as cost savings, independence from proprietary vendors, and improved security can also be mentioned. Additionally, developing economies of which have had problems adhering to intellectual property rights are on many occasions using open source software as a means to reduce this problem [Web03].

In Norway, open source software and open standards have been given significant funding in a recent national budget proposal [oA06], showing that the open source phenomenon has gained widespread acceptance here as well, causing an increased emphasis on technological openness in the Norwegian public sector.

3.10 Summary

Open source can be traced back to the early days of computing and the research environments. The evolution of open source has been heavily influenced by the improvements of network infrastructures and technical inventions. The evolution of the ARPANet, development of communication protocols, and development the Unix operating system by unpaid volunteers can be said to be prerequisites for the evolution of open source to what it is today.

The founding of the Free Software Foundation and the development of GNU, the emergence of Internet, and the development of the Linux Kernel mark the shift toward an organized free software development process. While the free software development process was proven successful in the case of Linux and the Apache http server, it was not until the founding of the Open Source Initiative and the release of the source code of the Netscape Navigator, that a large amount of commercial actors began adopting an open source strategy.

Today, open source is a development process in itself, that has gained a large amount of interest in both the general public media, by commercial actors, and national governments, deeply affecting the industrial dynamics of the software industry.

²²Open standards are publicly available and implementable specifications, not locked to any proprietary vendor

Chapter 4

Open Source Characteristics

This chapter provides a description of what is considered open source software, and its characteristics. This includes a definition of the term “open source” and what comprises an open source community. An overview the organizational structure and roles, of the rationale for individual participation in open source projects, as well as an explanation of the open source development process is provided.

4.1 Definition

The concept of open source is interpreted in different meanings depending on the context in which it is considered [WW01, GA04]. Some use the term about all software where the source code is publicly available. However, software distributed at no charge with the source code available can still be proprietary if, e.g., the license does not permit free redistribution and modification of the source code. Thus, proprietary software does not necessarily implicate that the source code is closed¹, nor is the availability of the source code equivalent with the software being open source. The visibility of the source code is not the issue, but rather what is allowed to do with it.

In addition to the ambiguity of what is meant by open source depending on the context, there are also several terms that are used that generally refer to the same principle. The Free Software Foundation uses the term “*Free Software*”, and provides a definition of what this entails. To be considered free software, the license must provide four freedoms: The freedom to run the software, the freedom to study and adapt the source code, the freedom to redistribute the software, and the freedom to improve the software and release the improvements to the public². The Open Source Initiative has provided their own definition of what criteria software must fulfill to be considered open source, containing 10 requirements³. These requirements include that it must be possible to freely redistribute the software, that the source code must be available for everyone, and that redistribution of modifications must be allowed. Generally, there are few differences between what comprises free software and what comprises open source software, notwithstanding the ideological differences of these two movements. Due to these reasons, terms such as “*Free/Open*

¹The software being distributed in binaries only

²Foundat:<http://www.gnu.org/philosophy/free-sw.html>

³The OSD can be found at: <http://www.opensource.org/docs/osd>

Source Software” (FOSS) and “*Free/Libre Open Source Software*” (FLOSS) are often found in the research literature, referring to software that is both free and open source. In this report, no subtle distinction relating to the terminology is made, and what some describe as Free/Open Source Software will herein be considered open source.

4.2 The Community

Much of the open source literature focus on open source development being performed in a highly collaborative environment, in which the developers are distributed across many different locations around the world [Mor05] [LW03]. Even though this might be argued not to be the case in all projects [ES06], the open source community comprises a wide variety of participants. Development is performed both by computer science students and professional developers as well as others with technical knowledge. According to a visitor survey performed by the Open Source Technology Group (OSTG)⁴ in March 2006, 95% of their visitors are men, with the average age of 32. About half of their visitors have a college degree or higher [ES06]. Similar data is found in [HO02], a study that also discovered that 58% of the total working hours were performed by professional developers, while the rest were students (14%) and hobbyists (28%). This can be related to a similar survey in which approximately 80% of the open source developers either were professionals or were studying in a related field [GGKR02]. The majority of developers were also found participating in more than one project [HO02], although the time spent on open source development was found to be generally low [GGKR02]. This study also found that the great majority of open source developers were from Western Europe or North America. When it comes to those that are not actively participating in the development efforts, due to, e.g., lack of technical knowledge, these users can in many cases provide important input to the project, such as bug reports, feature requests, and reporting usability concerns.

4.2.1 Individual Motivations

Economic theory tells us that a programmer participates in a project only if a net benefit can be derived from engaging in the activity [LT02]. Hence, with such a variety of developers, it may seem like a paradox that so many people contribute to open source projects without receiving direct economic compensation. Indeed, thousands of developers who do not receive any direct monetary compensation have succeeded in providing an enormous amount of code, supplying high quality and complex programs [Ben05].

This has led to much research being performed to identify why people participate in open source development efforts in their spare time, and a division into what has been called “extrinsic” and “intrinsic” motivations have been common [Kha00, BR03]. There are several specific motivations that have been identified in the literature, and an open source developer can typically relate to several of these [Kri02, HO02, LW03]:

⁴Owner of SourceForge.net, Slashdot.org, freshmeat.net and more

Participation in an Intellectually Stimulating Activity

Quite a few people find coding itself fascinating and want to participate in open source projects to challenge their intellect and come up with creative solutions to difficult problems. With this motivation, programming in itself is a hobby, and participation in open source projects is a way of enacting this hobby. Indeed, it was mentioned as the top reason for joining an open source project by [LW03].

Learning

Some open source participants join an open source project to achieve knowledge, either related to programming skills or how the open source community works, being mentioned as high as 70-80% in two open source surveys [GGKR02, HO02]. As we have already seen, many of the participants are students that use open source as a compliment to their studies, or directly related to school work. As the code quality found in open source projects often is high [DGMN02], this can help the to identify and learn good coding practice and style. Still, the mentioned studies comprised many professional developers describing the learning aspect as important, thereby making it a universal motivational factor.

Future Monetary Rewards

It has been shown that the core developers in high-profile projects receive higher salaries in their day-time job than other programmers [HO02][LPT06]. When someone participate in open source projects and provide good contributions, they both prove their skills to future managers and gain specific knowledge of these open source projects that outsiders do not possess. This causes a self-marketing effect; improvement of job opportunities was mentioned as an important motivator by 24% of the respondents in one survey [GGKR02]. In addition to proving ability by open source participation, it is also a possibility of being recruited by a commercial actor if they perceive someone as influential in a project in which they take interest. Future earnings can also be perceived by providing related products and services to the project that the contributor participates in, or by the networks established by such participation [HO02].

Belonging to a Community

The feeling of belonging to a community has been mentioned as a motivational driver [HO02, LW03]. Some find the participation in an unknown form of collaboration interesting [GGKR02], and thereby join open source communities based on, e.g., social gratification. When belonging to an open source community, the building and further enhancement of social and commercial networks is quite common. Additionally, by being a good developer in an open source community, one might gain a high reputation and a feeling of “power”. The rapid, constrictive feedback of contributions that is common in such communities, also work in a self-reinforcing way [HO02]. This can, e.g., increase the perceived self-worth and self-esteem.

Need of the Software

The need of the software has been described by Eric Raymond as “scratching an itch” [Ray02], and relates to solving a problem of which there are not any existing solutions. This can be a driving force for initiating an entirely new open source project. It can also be a motivating factor to join a project that has potential of fulfilling a persons needs. This can take form in providing a fix for an annoying bug that has been discovered, or personal customization of the software. These needs can either be in relation with software for personal use or work-related functionality. Mostly, it is professional developers that mentions this a reason to join an open source project [HO02].

Altruism

Some people have such strong beliefs in the free sharing of source code that they participate in projects based on altruistic reasoning. By increasing the welfare of others, one can increase self-esteem and feel more fulfilled as a person, thereby relating this to charity work performed in other arenas. Even though the whole motivation can be “giving to others”, there might also be underlying factors such as gaining similar services back from the community. This can be related to the concept of gift economy [BL01]. 30% of the respondents of a survey on motivations for participations mentioned opposition to proprietary software as a motivational factor, a view that can motivates into contributing to open source development by somewhat altruistic reasons [GGKR02].

4.2.2 Organizational Structure and Roles

Even though open source projects have a less formal organizational structure than what is typical in in-house development projects, certain organizational traits within the open source communities can be identified.

Due to the voluntary nature of participation, there are no formal programs for controlling an open source project, and no authoritative leaders that monitor the development such as can be found in traditional development organizations [YYSI00]. However, the organizational structure is still quite rigid, typically being organized in a hierarchical fashion. This hierarchy is often referred to as a meritocracy, in which status is gained according to the contributions participants make, and their merits within the community [BR03].

Figure 4.1 gives an overview of this meritocracy and the distinguishing roles in an open source project. Anyone can join an open source project, and these are often referred to as “joiners” or “newcomers” [BR03]. The passive users are those that use the software, but do not participate actively by providing anything back to the community. People who are new to the project, but are participating actively are deemed of higher value to the project. Such participation can consist of reporting bugs, providing fixes for bugs, or even providing source code that implements new features. These contributions are then reviewed by the other developers and added to the code base if found worthy. By providing several high quality contributions, one can rise further and gain a developer status, which can include the rights to add source code to the repository and accept contributions from others. The core developers are typically a small group, consisting of the project initiator(s) and others that have proven invaluable to the success of the project. The meritocratic structuring means that the hierarchy is not strictly assigned from the beginning, but is evolving as

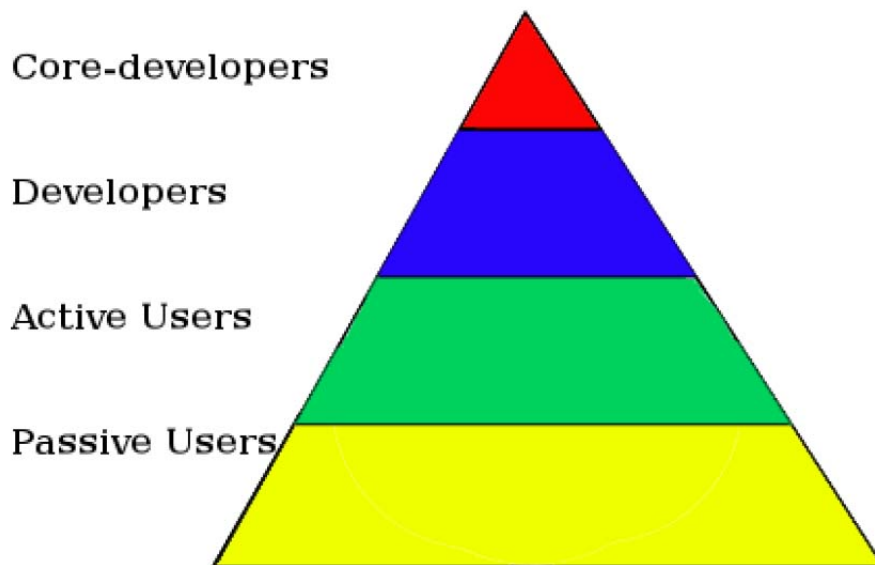


Figure 4.1: Roles in an open source project [HR06]

the time goes by. However, the initiator(s) will generally stay in the top of this hierarchy as long as they remain active in the project.

Even though the hierarchical authority often relates to actual decision making power, it is mostly related to how much people listen to your opinions [BR03]. It should also be noted that this authority only is valid as long as the rest of the community agrees on their legitimacy [Sca04]. If, e.g., the leadership is not accepted by the other participants, they may abandon the project or bring the code base along in order to create a new project with new leadership⁵.

4.3 The Development Processes

This section describes the typical traits of those open source projects that are initiated and is evolved by the open source community. Note that the process outlined here is not linear, and most of these activities will often be performed simultaneously, as well as following an iterative life-cycle.

4.3.1 Conceptualization

Open source projects typically start with the unfolding of an idea, being either in the mind of a single person or as a group effort. This will often relate to fixing a perceived problem, such as an identified lack of some software that the person(s) would like to have. The project initiator(s) will generally become the owner of the project, taking on responsibility of managing and organizing the

⁵Such a move is usually referred to as “Forking”, and can also be due to other reasons such as disagreement related to how to project should evolve

project into (hopefully) a successful one [vHvK03].

4.3.2 Requirement Specification, Planning, Analysis and Design

As the conceptual idea is invented by a single person or group, there is typically no formal phase of requirement specification in open source projects. The requirements are often taken as generally understood, since developers are also often the users of the software⁶ [Fit06].

4.3.3 Implementation

When the initiator(s) have coded a prototype based on the idea for an open source project, it is put forth on the Internet to attract interest from external people. Once the source code is publicly available, the initiator can then announce it over various news channels to attract interested people to the project. Hopefully, some will get involved by using the software and later contributing with fixes to defects, new features and generally helping with the continuous progress of the project. The implementation phase in open source projects is thereby highly iterative, with new modifications and new features implemented at a fast pace [vHvK03].

4.3.4 Writing Code and Submitting it to the Community for Review

When someone has created a contribution that is believed to be of interest to the community, it will be submitted to the community for review. Due to the nature of open source projects and the large extent to which one chooses what work to perform, these contributions are typically in accordance with individual expertise and interests of the contributor. The contribution then goes into a review process, which is of high importance to avoid having faulty code integrated into the code repository. This process can vary from the somewhat informal posting of the code on a mailing list, and adding it to repository if no one objects within a certain amount of time, to an involved process requiring authorities to accept the code, as is the case in the Mozilla project [RdMF02]. Often, the project leaders will request certain features, and will then receive several submits sent for review. The best solution is then chosen, typically based on a voting system.

4.3.5 Pre-Commit Test

It is important to ensure that a contribution is tested properly to not break the build. Projects can include many testing activities, including ad-hoc volunteer testing, smoke tests⁷, and even contributed functional tests [RdMF02]. For mature projects, an involved smoke testing procedure is important to ensure stability for all supported platforms. If the tests succeed, the source code is committed to the central repository. The actual committing of source code will in most cases be done by a few trusted developers, who serve as “gate keepers” for the code submitted as contributions [vKSL03].

⁶At least on a macro level, this is not necessarily true for all open source projects

⁷A series of tests to ensure that the software seems to work as it should, i.e., build verification

4.3.6 Development Release

Typically, the open source projects incorporate rapid implementations with frequent releases. These are the so-called development releases. Projects will then, e.g., conduct a nightly build in which accepted contributions are included. The possible large number of potential debuggers on different platforms and system configurations can then help to discover bugs and defects quickly in these releases⁸. This can be related to the so-called Linus' Law: "Given enough eyes, all bugs are shallow" [Ray02].

4.3.7 Production release

Many community-initiated open source projects do not follow a strict deadline in which the product must be finished. This usually causes the development to last for as long as the project authorities deem necessary, before a release is conducted. The production release is thus typically performed when a thoroughly tested, stable version of the product is "finished".

4.4 Summary

This chapter has explained what open source is, some typical traits of open source communities, including organizational structures and roles, as well as how the development generally proceeds in open source projects. It is important to have a general knowledge of these issues when considering to release an open source project. Knowing the distinction between free and open source software makes it easier to target the project better toward specific community groups, while knowing the motivations behind participation in open source development efforts can make it easier to present the project so that these motivations can be met. Knowledge of the organizational structure, and of the development process can aid project initiators to act as better leaders, and to better understand the social dynamics of open source projects.

⁸Although, it should be taken note that there are only a few developers in most projects [Kri02, MFT].

Chapter 5

Licenses and Legal Issues

Open source software unlike, e.g., software in the public domain¹, is copyrighted and distributed with license terms. These licenses mark a fundamental legal innovation, being designed to ensure that the source code will always be available. This chapter describes the most important terminology in relation with intellectual property and provides an overview of open source licenses in particular.

5.1 Intellectual Property

To understand how open source works, an understanding of intellectual property is required. Intellectual property can be defined as creations of mind; creative works or ideas embodied in a form that can be shared or can enable others to recreate, emulate, or manufacture them². Intellectual property can be protected by the means of patents, trademarks, and copyrights.

5.1.1 Patents

A patent can be granted for new, useful, and non-obvious inventions, and gives the patent holder the right to prevent others from using the invention without a license. Patent applications must be filed to each country the patent should act as a protection, and they are only valid in the countries where they have been granted. The sub-category of software patents is an important issue in relation with open source.

Software Patents

Software patents disallow the implementation of a certain idea into software. If an open source project is accused of infringing a patent, the project must either stop implementing- and remove the patented technology, or potentially face an expensive and time-consuming lawsuit³. The software patents are often invisible to end users, but developers must often pay licensing fees to develop

¹Software with no copyright holder

²According to the United States Trademark and Patent Office: <http://www.uspto.gov/>

³It is also possible to pay for a patent license, but this is usually not feasible in open source projects

software using patented technologies. Projects can be sued even though the software is not distributed for profit, posing a threat to the open source software community⁴. This might cause open source programmers to avoid patented algorithms even though they are the best or only solution. It might also be difficult to ensure that no patents are used in the software, as open source software often is the compilation of code from many sources [RE04].

5.1.2 Trademarks

A trademark is a distinctive sign that identifies and distinguishes a product, a service or an organization. A registered trademark will be protected in most industrialized countries, due to the Madrid System for International Registration of Marks⁵. If it turns out that someone has trademark protected, e.g., the name or logo used in an open source project, these much be changed to ensure the avoidance of legal repercussions.

5.1.3 Copyright

A copyright protects an original artistic or literary work. It only covers the form that have been manifested in a material expression, and in relation with software this applies to the source code. The copyright gives its holder the right to control the reproduction or adaption of the works for a certain amount of time. Copyright protections are valid in countries that are members of the World Trade Organization, being standardized through the Berne Convention. Hence, this protection is applicable nearly worldwide. Use of the copyright symbol: ©, is not required for the copyright to be in effect. Indeed, all source code is automatically copyrighted unless released into the public domain. If part of the code looks like it may infringe on someone else's copyright, that part may be rewritten to avoid breaching the copyright.

5.1.4 Software Licenses

A license enables an entity that owns property rights⁶ in something, such as the copyright, to grant to a third party⁷ the right to use those property rights. [CTW98]. A software license can thus be described as an agreement between the creator and the user of some computer software, comprising the terms for the use, modification, and distribution of the software. Any third party who uses the software without a license infringes on the owner's intellectual property rights [CTW98]. The copyright holder might choose to license the software with several licensing schemes according to how the software is to be used. This strategy is used in relation with the dual licensing model that some open source vendors use⁸.

⁴Lawsuits for software patents are generally only a problem in the USA. E.g., the EU does not allow for patenting of software innovations: <http://www.european-patent-office.org/legal/epc/e/ar52.html>

⁵This applies as long as the countries do not refuse protection of the designated mark. A full listing of countries where this protection apply can be found at http://www.wipo.int/treaties/en/documents/pdf/madrid_marks.pdf

⁶The Licensor

⁷The Licensee

⁸The dual licensing model is explained in Chapter 6

5.2 Open Source Licenses

Open source licenses grant a variety of rights as described in the open source definition, including the right to modify and redistribute the licensed software, something that otherwise would be forbidden by copyright law. There exists a large amount of different open source licenses⁹. Most of these are based on the earliest ones, such as the GNU General Public License, the GNU Lesser General Public License and the Berkeley Software Distribution License(s). This section provides a brief overview of these three licenses, which are also the most commonly used ones [dM03, LT05b] as well as the Apache License, the Mozilla License, and the Eclipse Public License due to the influential position these license have in the community. Most open source licenses include some terminology and exhibit common properties¹⁰ that will be described before going into the details of each license.

5.2.1 Terminology and Common Properties

Derivative Works

The term “derivative works”, is commonly used pertaining to open source licenses. It is an important issue since some licenses require the publication of the source code of derivative works. It can generally be said that derivative work is work that uses or extends the work of someone else, i.e., by using or modifying the source code of some software. There is, however, some controversy and different views on the definition of derivative works. This can, e.g., be related to details such as how extensions are linked with existing software, and the creation of dedicated licensing schemes might in such instances be recommended [RE04].

Copyleft

Copyleft is the wording given by Richard Stallman on the use of copyright law to achieve a result opposite of traditional copyright. When a license uses the copyleft-concept, all modifications and derived works of the licensed software must be redistributed under the same license. Thus, this mechanism is highly protective of the creator of the project, in that it helps ensuring the avoidance of commercial takeover of the source code. Copyleft is also often referred to as the “viral effect” by critics, as modified and derived works can be said to be “infected” with the original license.

License Compatibility

License compatibility refers to the concept of being able to combine source code licensed with a variety of licenses to create new work. If the licenses contain contradictory requirements, these licenses are incompatible, and the source code cannot be combined. Most open source licenses are compatible with each other as well as with proprietary software licenses; the major exception being

⁹A full listing of the licenses accepted by the Open Source Initiative can be found at <http://www.opensource.org/licenses/alphabetical>, while the Free Software Foundation has its listing at <http://www.fsf.org/licensing/licenses/>

¹⁰In addition to the adherence of the OSD

GPL. It is important to ensure that all of the licenses are compatible if the developed software is going to be distributed or sold to the general market [CLM⁺07].

Copyright Notice

Most open source licenses include a clause stating that a copyright notice and intellectual property used in the contributions must be made available to everyone.

Liability Disclaimer

The absolute majority of open source licenses include a part disclaiming all forms of responsibility for liability and warranties in relation with the licensed software, unless otherwise stated in separate licensing schemes¹¹.

Patent Retaliation Clause

To protect against patent-related lawsuits, most of the newer open source licenses include some form of patent retaliation clause. These measures might, e.g., as is the case with the Apache 2.0 License, ensure that the licensee's rights, such as the right to redistribute the software is terminated if it is attempted to enforce a patent under the circumstances specified in the license.

5.2.2 GNU General Public License (GPL v.2)

The GPL license was introduced by the Free Software Foundation, and is today the most commonly used open source license¹². The General Public License was first written by Richard Stallman in 1989, but was superseded by GPL v.2 two years later¹³ [Ric91]. The primary goal of GPL in accordance with the philosophy of the FSF, is to promote free software. The license hence deliberately written to ensure that GPL-ed code cannot be mixed with proprietary software. This is ensured by the use of the copyleft property, imposing that all derivative works must be distributed under GPL, and that no additional restrictions can be placed on the redistribution of the original and derivative works¹⁴. In the case of GPL, this means that all derivative works must be distributed in source code along with the binaries, efficiently making it impossible to incorporate a GPL-program into proprietary software for closed source commercial distribution. However, the license does allow anyone to charge a price for this distribution (although in practice such fees are generally very low), as well as to provide warranty protection in exchange for a fee.

¹¹To allow for the sale of liability and warranty services

¹²This can be seen, i.e., in the statistics at Freshmeat <http://freshmeat.net/stats/>

¹³At the time of this writing, a third version of GPL is forthcoming.

¹⁴It is allowed to create derivative works for personal or organization-internal use, meaning that this clause only relates to distribution

5.2.3 GNU Lesser General Public License v.2.1(LGPL)

LGPL was introduced alongside GPL v.2¹⁵ and although being less restrictive than GPL, it generally impose the same requirements. It differs from GPL in that it permits the linking of libraries into non-free programs [Sta99]:

“A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a “work that uses the Library”. Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License”.

The Free Software Foundation has stated that the reasoning for the introduction of this less restrictive version of the GPL was to encourage a larger user base for certain parts of the GNU libraries, e.g., in order for a library to gain status as de-facto standard. However, if someone makes changes directly to LGPL-licensed libraries, the license will come into effect. This includes that:

- The modified work must itself be a software library.
- The files modified must carry prominent notices stating that the files have been changed and the date of the change.
- The whole work must be licensed at no charge to all third parties under the terms of the license.

5.2.4 Berkeley Software Distribution License(BSD)

The BSD license was written at the University of California at Berkeley for use with their early Unix Distributions. This original version of the license included an advertising requirement, in which an acknowledgment to the original source was required¹⁶. The requirement became unpopular by many and also made the license incompatible with GPL. After receiving a letter from Richard Stallman, it was officially rescinded in 1999 [Uni99], leading to a rewrite of the license to what is now known as the Revised BSD^{17,18,19}

The BSD License is one of the less restrictive open source licenses, as it grants the right to the use of source code and binaries in every possible way. This includes the right to create a proprietary product for commercial distribution where the source code is not available²⁰. The only restriction to speak of in relation with the Revised BSD is that the copyright notice that cannot be used to “endorse or promote products [...] without prior written permission” [Uni99].

¹⁵It was initially released as the “Library General Public License”, later changing the name to “Lesser General Public License”

¹⁶Which in time could evolve to a significant amount of “sources”, becoming a highly impractical requirement in projects with a high number of contributors

¹⁷Making the BSD essentially similar to the MIT/X license, and making the BSD compatible with GPL

¹⁸This license also goes by the name of the New BSD

¹⁹The Original BSD is still in use, the freeBSD project being an example. Both FSF and OSI discourage the use of this old license

²⁰Several commercial actors have implemented BSD-licensed software in their solutions, e.g., Microsoft has publicly admitted the usage of BSD-code in their Windows operating system

5.2.5 Apache License

The Apache License is well known, and quite widely adopted due to its use in the Apache applications [ES06]. A licensee of Apache licensed software can copy, modify and distribute the software in source and/or binary form, working much in the same way as BSD license. The license has a distinguishing feature in which adds an explicit grant of patent rights [The04]. This also includes a termination clause in which the license rights are revoked if the licensee sues over patent infringement²¹.

5.2.6 Mozilla Public License(MPL)

The Mozilla Public License (MPL) was created by Netscape as a part of the release process of the source code for their Netscape Communicator browser, and is today mostly used with software in the Mozilla Foundation. This license explicitly permits MPL-licensed code to be combined with separate proprietary source code to create a proprietary work for further sales [Moz98]. Still, if any modifications are made to the code itself, these modifications will be affected by the license. It can be said that MPL in this regard is somewhat influenced by GPL, meaning that to avoid source code to be affected by the license, it must be kept in separate files from the MPL code²². If these demands are not fulfilled, the source code must be made freely and publicly available just as is the case with the GNU Licenses. Additionally, MPL grants the right to use any IP associated with the source code, leading to the requirement that all contributors must grant the rights to any intellectual property that is used in their contributions.

5.2.7 Eclipse Public License(EPL)

The Eclipse Public License (EPL) was created in conjunction with the forming of the Eclipse Foundation, and all Eclipse projects are licensed under this license [Fou07b]. It is based upon the former Common Public License (CPL). As the CPL that is essentially the same as the MPL [GG05], EPL and MPL are very similar. The major difference is that whereas the MPL differentiates at changes to the source files, EPL allows for the creation of separate modules under their own license agreements, as long as these are not derivative works.

5.3 Summary

This chapter has described the concept of intellectual property and its protection mechanisms. It has provided an overview of how these protection mechanisms relate to software distribution, and open source software in particular. The concept of open source licensing and some common properties of such licenses has been described. In addition, a more detailed description has been provided of GNU GPL, GNU LGPL, the Berkeley Software Distribution license, the Apache License, the Mozilla Public License, and the Eclipse Public License. It is important for commercial actors to have knowledge of the concepts presented herein when considering to release software as open

²¹This makes the Apache License incompatible with the GPL(v2).

²²This is by some referred to as being weakly copyleft.

	Copyleft	Include in proprietary	GPL compatible
GPL	Yes	No	Yes
LGPL	Weakly*	When linked	Yes
BSD	No	Yes	Yes
Apache	No	Yes	No**
MPL	Weakly*	Separated source	No
EPL	Weakly*	Yes	No

Table 5.1: Properties of some of the most used open source licenses.

source. Knowledge of intellectual property right protections and open source licenses, can help making sure that lawsuits are avoided. It is also important to ensure that the licensing choice is fitting for the project and its surroundings, and that it is suitable in accordance with how the company intends to derive benefits from adopting an open source strategy. These issues are further explained in Chapter ??.

The most important properties of these licenses are summarized in Table 5.1:

*LGPL: Linking to libraries of separate work is allowed

MPL: The new separate work must be kept in separate source files from the MPL code to avoid copyleft

EPL: The new work must be kept in separate modules from the EPL code to avoid copyleft

*Apache License 2.0 is compatible with the upcoming GPLv3

Chapter 6

Commercial Adoption of Open Source

Open source strategies are increasingly being adopted by commercial actors [Fit06], and it is agreed in the literature that cooperatively developed software has acquired economic importance [Bär06b]. There are several potential benefits that can be associated with adopting an open source strategy in the business environment, and it has been reported that the use of open source software has resulted in significant savings in some organizations [Fit04]. This chapter describes aspects of open source strategies that are related to commercial use and development practices. Additionally, it will provide a brief description of some business models that can be used in open source projects, that is, how commercial companies can increase their revenue by using open source strategies.

6.1 Commercial Use of Open Source Software

The commercial use of open source software, the potential benefits as well as pitfalls to be aware of, are described in this section.

6.1.1 End-Use of Open Source Products

There is a significant increase in the use of open source software in the commercial sector [Gho06]. Benefits regarding the end-use of open source products mainly relate to increased savings, i.e., not having to pay licensing fees. Much open source software provide good implementations, making it unnecessary to spend money on commercial software. Indeed, open source software has in many cases proven equal to- or arguably exceeding commercial alternatives considering quality, reliability, and security [RE04], making such software the natural choice. Open source is also considered to provide better interoperability due to its association with open standards. There are no vendors that try to protect their market-share by locking the software to proprietary standards.

The main arguments against using open source from an end-users perspective, relate to low usability¹ and lack of support often found in open source software [NT03, LT02]. Still, such issues seem to be given more considerations since open source software is increasingly entering the commercial

¹especially for the non-technical user

mainstream [JF00], possibly reducing the validity of such an argument. Additionally, considering the widespread use of commercial software such as Microsoft Word and Microsoft Excel, the exchange of documents between proprietary and open source solutions can be troublesome due to the adherence to a proprietary format in the former solutions. This will probably be less of a problem in the future as, e.g., Microsoft has taken recent initiatives to expand document interoperability². It should be mentioned that the use of open source tools in development activities might in exceptional circumstances, cause the resulting artifacts to be considered derivative work, thereby giving the copyright holder certain rights [RE04].

6.1.2 Reuse of Existing Components

Most open source software is obtained from external sources [dM03], and the reuse of existing components is probably the most common way to take advantage of open source in software development³. With this approach, most of the work has already been performed by others, hence resulting in less internal development efforts. This can especially relate to less time spent on common programming tasks and easier adoption of new technologies. The use external workforces makes an effective use of specialists and has been said to reduce the process risk as well as providing increased reliability [ABNR05].

Even though the use of external components can provide reduced costs and time to market while at the same time providing products of increased quality, it must be emphasized that this approach also brings along certain problems and risks. Much efforts can be spent trying to find suitable components where none exists. and even when suitable components are identified, correctly choosing one component over another can prove a difficult task. The use of off-the-shelf components will also in many cases require company-specific adaptations, possibly causing increased maintenance costs. It might in these cases be considered whether to freeze the component as-is. Such a strategy will however make the company unable to take advantage of new releases. By staying updated with each new version, the maintenance work might in some cases prove a heavy burden. The lack of documentation, testing and field support in many open source projects [LCS⁺05], can also cause problems in initially getting to know the source code, thereby making the learning curve steep. This will in turn cause additional indirect costs. Because of the problems mentioned, companies should not consider open source as gratis software, as it certainly can require substantial investment before it can be deployed into the marketplace [RE04]. Indeed, in some cases it can be more worthwhile to buy a component over choosing a similar open solution, due to reduced total cost of ownership.

6.2 Commercial Participation in Open Source Projects

It has been noted that many firms are profiting from OSS, both large companies that allocate part of their intellectual property to the community and derive benefits from the interactions with the community, as well as small software houses adopting a business model centered on open source software [BR03] [DM05] [BR06a]. Some companies also release software that is only used internally, to gain contributions from the community and thus improved product quality [Fit06]. Companies'

²A press release can be found at: <http://www.microsoft.com/presspass/press/2006/jul06/07-06OpenSourceProjectPR.mspx>

³These components are often referred to as Off-The-Shelf(OTS) components

participation in open source projects can be divided into two kinds of participation; collaboration in development efforts of external projects, and the release of internally developed source code, i.e., the initiation of an open source project.

6.2.1 Collaboration in External Projects

Community founded open source projects are initiated by one or more individuals independent by their employment context [WM05]. These projects typically remain community-managed, and are usually structured to prevent a takeover by a firm or another organization [WM05]. The opportunity is still present for companies to sponsor the development of such projects, and this approach has been taken by a large amount of commercial firms, i.e., in relation with software of the Eclipse Foundation and the Apache Foundation. By paying employees or external developers to create some functionality or to fix bugs in the software, companies can ensure that their needs are met, much similar to the individual motivation described in Chapter 4 pertaining to a commercial actor. Sponsored contributors must in these cases earn and sustain their role in the project under the same terms as the volunteers. An additional motivation of commercial collaboration with external open source projects relates to the increased visibility, and the marketing effect this creates, as well as the possibility to create plug-ins for certain products that can be licensed and sold. Helping with the development of external open source projects might also be a measure to reduce the market share of a large competitor, by helping to create a competitive open source alternative. West and Mahony suggest three reasons as to why sponsors would initiate an open source project. Firstly, they want to create a larger market for the project's software or reduce a competitor's market share by building a robust development ecology. Secondly, the use of external contributors reduces the demand on the sponsor's resources so they can redeployed. Thirdly, the sponsor hopes to create software as a public good - particularly likely for a nonprofit or government sponsor [WM05].

6.2.2 Releasing Code as Open Source

At the first glance, open source exhibits a number of features that make it rather unattractive as a mode of production for commercially motivated producers. It makes commercial licensing unfeasible since the open source licenses imply that no restrictions on further dissemination may be asked by any recipient. The conventional business model of most commercial software producers is vitally based on such restrictions [Bär06b]. Even though the release of software as open source may initially seem paradoxical going against common economic traditions, there are certainly benefits associated with such a strategy. There have been identified two major reasons as to why commercial companies release their source code to form an open source project; to gain development assistance on non-critical areas, and to win adoption [Wes03].

6.2.3 Development Assistance

The possibility of contributions from external developers seems like an obvious benefit. If the community takes interest in the project, a plethora of skilled developers might provide detailed bug-fixes, security-fixes, and even new features for the project, improving the value of the final product. The peer-review will in turn also help making the product better, more reliable, and more

secure [Whe05, RE04]. In addition to direct contributions to the software, development of plug-ins and add-ons might take form, which in turn increases the total value of the product. These can also be provided from the company using normal license sales (as long as license regulations are adhered).

6.2.4 Increased Adoption

Most scholars have thus far argued that in the face of strong incumbent monopolies and intellectual rights regimes favoring bigger competitors, open source might act as a counter strategy against the monopolies leading to increased adoption of the non-monopolized products. Netscape made most of its money selling server software and for them, the issue was less about browser-related income than maintaining a safe space for their much more valuable server business [Kri05]. If Microsoft's Internet Explorer achieved market dominance, Microsoft could be able to bend the Web's protocols away from open standards and into proprietary channels that only Microsoft's servers would be able to service [Kri05]. Thus, Netscape released the source code of Netscape Navigator to counteract the monopoly of Microsoft Internet Explorer, a goal that can be said to have been achieved. When the product is given away free of charge, it may gain a highly increased user-base, possibly increasing the popularity of the company. The company may thereby derive additional benefits related to their other products and services, as well as reducing the market share of competitors. An increased user-base will also cause the program to be tried out on a higher amount of platforms and hardware configurations, revealing compatibility problems and other issues that must be addressed.

Negative aspects with such a move mainly relates to the lack of licensing-revenues from the customers, although this can be taken somewhat into account by providing hybrid licensing schemes. In addition, there will never be any guarantee that a large mass of people will take interest in the project. If this does not happen, the anticipated benefits cannot be derived, while at the same time the code is left open for everyone.

6.3 Earning Money from Open Source

As opposed to the more traditional way of developing and selling proprietary software, open source software sets the stage for generating income by other means. The commercial entry into the open source arena has led to the emergence of new business models. These models for generating income are typically based on providing value-added services, although direct license sales can be a coexisting possibility. We will below explain some of the business models that are applicable in relation with open source software⁴ [Ray02, Hec00, Fit06]:

6.3.1 Support Sellers

In this model, the application is given away for free and revenue is generated by other means. The revenue can be based on distribution, branding, and after-sale services, and has been successfully accomplished by well-known open source companies such as Red Hat and eZ Systems. To be

⁴It is common to adopt several of these business models in combination

successful with this business model, it is important to differentiate from the competitors. This can be done by providing better and easier-to-use distributions than competitors, or by providing similar services at lower costs. In relation to this, the first mover advantage should be mentioned. This means that the first company that manages to attract a large customer base will be difficult to out-compete, something that has been an important factor in the success of, e.g., MySQL and eZ Systems. It is also possible to provide services such as support, warranties, and sale of liabilities to generate income.

6.3.2 Loss Leader

In this business model, an open source product is offered free of charge with the intention that this product will increase the sale of other company products that are following a traditional business model as exemplified by, e.g. Netscape. This can work in several ways. One example is that the open source product is used to build the brand and reputation of a company. By releasing the product freely, the user base of the product can be increased, hopefully leading to increased future sales of other products. Another approach is to release software for free that complements other software, leading to increased value of these products which are sold under commercial licenses.

6.3.3 Widget Frosting

This business model is a likely choice for companies where the revenue mainly is generated through the sale of hardware. By supplying open source software such as drivers, complete applications or even operating systems that support the hardware, this can lead to increased sale of the main product, hardware. This approach has been adopted by companies such as Apple, IBM, Sun and Novell.

6.3.4 Accessorizing

This business model is intended for companies that distribute books and other accessories associated with open source software, such as O'Reilly. For example, companies that are selling books explaining how to program in Perl or how to setup Linux distributions are following this business model. Since the software is licensed as open source, this also means that the software can be bundled together with the book adding to the book's value at a low cost.

6.3.5 Hybrid Model

The hybrid model involves software distributed under licenses that are not considered true open source, nor as restrictive as traditional proprietary licenses [Hec00]. The idea is to take advantage of open source while still being able to create revenues directly from the sale of licenses. This is generally accomplished by either limiting the availability of the source code or by treating the users differently according to the context in which the software is to be used. This is often referred to as dual licensing, a concept that has been successfully adopted by several companies such as Trolltech and MySQL. Dual licensing can be done by providing one restrictive open source license

such as GPL that makes the software free of charge to everyone, as well as a commercial license that must be bought to include the software in other commercial products. Then the company is able to benefit from the traditional IPRs-based source of revenues, while still releasing open source software to the community, i.e., increasing the adoption of the product, making the company better known, and the potential for receiving bug-fixes and contributions from the community.

6.3.6 Leveraging The Community

This business model relates to the release of a software product to get contributions from the community. Leveraging the open source community allows companies to increase productivity, and to increase the value of the product. While this is an effect hoped for in all open source projects, it can still be considered as an open source business model, due to it being the sole reason as to why some companies release their internal products as open source. This can typically relate to consultancy companies that develop software for use in their consultancy projects, and being non-critical to their business. They release these products as open source hoping to get help in the development efforts from the open source community, making the software better for their own use.

6.4 Summary

This chapter has presented some arguments for and against commercial adoption of open source, relating the arguments to use, reuse and commercial participation in open source projects. It has also been described how companies can earn money on open source, i.e., by providing a description of open source business models. Several of these business models are typically combined to form the open source business strategy. A working knowledge on how and why companies approach an open source strategy, its benefits, and the potential negative aspects are important background knowledge for companies considering to adopt an open source strategy. This will further be elaborated in Chapter 7.

Part III

The Release Process of Open Source Projects

Chapter 7

Identifying the Rationale

This chapter addresses research question 1.1, and focuses on identifying the rationale for commercial approaches to open source software. It has earlier been shown how open source has evolved to become a new separate development methodology for commercial companies, and the previous chapter outlined aspects related to commercial adoption of open source strategies, as well as business models that can be adopted to create revenues off an open source strategy. Based on interviews with four commercial actors, and from the involvement in the release of an open source project, this chapter outlines the first step to consider in relation to adoption of such a strategy; identifying the rationale.

7.1 Motivations

There are in general two major motivations behind the release of open source software; to increase the value of the product, i.e., for internal use, and to achieve an increased revenue stream.

7.1.1 Increasing the Value of the Product

It is important to have a pragmatical view on open source, and an open source strategy should only be adopted if the benefits of such a strategy outweighs the potential negative consequences. Bekk¹ is a Norwegian consultancy company, that uses and develops open source software for pragmatical reasons, to ensure the provision of effective consultancy services. Stefan Landro explained this issue in the following way: *“Bekk has been positive to open source for a long time. We use open source where it is meaningful. For us, it isn’t about evangelizing, but rather to provide what is best for the customer. In many cases what is best for the customer is open source - we have a very pragmatical view on it”*.

He then goes on explaining the reasoning behind their open source approach: *“We do not sell software, we sell services. We sell consultancy services based on people being good at what they do. In other words, we don’t pay our employees to develop open source in the way, e.g., JBoss*

¹Bekk and the interviewees, Stefan Landro and Christian Schwartz representing the company, are presented in Appendix B.

does. *We create tools to make our own work easier, tools that can be reused in many projects. That is our motivation, we have a need for the software. We use open source in many projects, and we often contribute by solving the problems we experience. The advantage with this model is that others benefit from our contributions and we benefit from theirs. Also, if we need, e.g., a tool for use in several projects, and there aren't any alternatives on the market, neither open source nor commercial, then we might create an open source project ourselves*".

The context, the profile of the company is important to consider in relation with motivations for approaching open source software. Companies might consider to release the source code as an open source project of software that they use internally in their company to achieve a better product, i.e., to get contributions from the community. As we already have seen, Bekk uses this approach. Keymind is a small Norwegian company of which the release process of their monitoring software, Keywatch, has been followed closely. They are motivated for much of the same reasons as Bekk, as they were already using Keywatch in their daily business when it was released as open source. The software is able to monitor a large amount of processes, and is developed with a modular architecture that strongly supports the creation of plug-ins. Their main goal from releasing the software as open source was to gain additional plug-ins from the community, and Audun Jensvoll described their reasoning to release it as open source in the following way: *"By offering Keywatch under the Apache 2.0 License, we hope to build a community that will contribute functionality for Keywatch to support common monitoring challenges and a lot of specific ones, making Keywatch an attractive alternative to expensive and less flexible systems"*.

Linpro² is a Norwegian company that derives their main revenues from open source services, and they have taken a similar approach to Bekk and Keymind, as explained by Thomas Malt: *"We have some projects that we develop internally, and just release as open source to see if anything happens"*. He goes on explaining how their main project, Varnish came to be: *"What happened with Varnish was very interesting. We were contacted by VG³ with an idea for an open source project. They wanted us to organize the project in its entirety. We have a couple of people that are very involved in the BSD community. One of these knew a person from Denmark that was perfectly suited for helping with the development of Varnish. He was contacted, hired, and the project was organized in cooperation with VG"*.

Christian Schwartz also had some input to explain some of the reasoning behind their release of open source products. *"It is more easy to be taken seriously if the project is open source. People are skeptical to products that cost money, especially related to software developed in Java. To successfully sell software developed in Java, it must be something like server software, profiling tools, a database, or something that solves a huge problem. People are often not willing to pay for something they don't have much knowledge of"*.

Stefan Landro explains the importance of competency building, and how open source can aid in such a process: *"Building of competency is an important part of being a consultant. Bekk notices those that are active in open source projects. We create value for the company by creating our own open source projects in the sense that we add to our knowledge, i.e., increased competency. We use the software ourselves and save time by using the software. By developing the products ourselves, we also become very good at using these products"*. Christian Schwartz emphasizes some of the individual motivations that was identified in Chapter 4: *"You have the concept that partaking in*

²Linpro and the interviewee representing the company, Thomas Malt, are presented in Appendix B.

³A Norwegian news provider

an open source project is a great way to become a better programmer, and that it is fun”.

The common denominator of the cases listed above is that the software that is released as open source is not critical to the main business in the companies. They use the software for internal purposes, and hope to achieve a better product by releasing them as open source. The release of open source software in these cases can also help to profile the company in a beneficial way, they might receive a “free” advertising effect, making the company’s name better known to the outside world.

7.1.2 Creating Revenues from Open Source

While the release of software as open source can be a successful strategy to obtain contributions from the community, it is still important that the company carefully evaluates the open source strategy up against the use of a proprietary one. Generally, it can be said that the main driver for a commercial actor is to increase its revenues. Therefore, proprietary software should only be released as open source in the case that an economical effect can be obtained either directly or indirectly. This can, as we have seen, relate to increasing the income of the company due to more effective internal use. It can also relate to increasing the company’s current or future earnings as well as decreasing the market share of a competitor, making the total monetary gain higher than what would have been the case if the software had not been released under an open license. Actually measuring these gains may prove difficult, especially when related to indirect effects such as adversely affecting a competitor or the resulting advertising effect that leads to higher revenues from other products.

Most open source companies calibrate their degree of openness and maintain a mixed product and service portfolio [BR06a]. This is the approach taken eZ Systems and Trolltech, two companies that can be said to base their strategies entirely on their open source solutions.

The founders of eZ Systems⁴ had been interested in open source for a long time. They started out by developing a commercial solution, while at the same time selling consultancy services in their nearby region. Vidar Langseid, the head of development explains how eZ Publish came to be, and how they ended up adopting an open source strategy: *“In the beginning, we developed a commercial product for the stock market. To create income while this software was under development, we sold consulting services in the nearby region, much of which consisted of creating web sites for our customers. Instead of creating each web site from the scratch, we developed a framework that was common for each customer. This framework evolved to what is today known as eZ Publish. It was released as open source due to our ideologies, while we at the same time created the proprietary software for the stock exchange market. eZ Publish gained a large amount of interest and we decided to work on the open source project full time, setting the commercial software aside”.* They decided upon a business model related to the selling of services, defined as the support seller model in Chapter 6.

Trolltech⁵ is a company that bases their income⁵ on the hybrid dual licensing model explained in Chapter 4, and they have become known worldwide due to their Qt software. Knut Yrvin explained how their business model works in the following way: *“Trolltech’s business model is visible through*

⁴eZ Systems and the interviewee representing the company, Vidar Langseid, are presented in Appendix B.

⁵Trolltech and the interviewee representing the company, Knut Yrvin, are presented in Appendix B.

what we call free software. Trolltech also has a proprietary license so that Qt is dual licensed. This means that people pay us to use the software as a development tool to create their own programs. However, if they pay for the proprietary license they do not have to send the source code back. If they use the free version of Qt, they have to give their source code back to the community. That's the main rule". He continues: "The advantage of the dual licensing strategy is that by being commercially successful, although some people will argue against this [...], make us able to contribute more to free software. It is a win-win situation". Certainly, Trolltech can also be said to use their open source model to achieve more customers for their proprietary version, i.e., the loss leader model: "People have used Qt as an open source product, and brought it along to their workplace, and people have learned how to use Qt in their studies, and have brought it along to their workplace". When Qt is to be used commercially, a license fee must be paid and thus, the open source version acts as a loss leader. Trolltech also increase the value of their product by using an open source license: "Very many use Qt, our main product, and about 50 percent of the feedback we receive comes from the open source community[...] This feedback helps us to deliver an industrial product notably faster than our competitors. In other words, the open source community helps us to make our product better".

Even though Trolltech has been successful in their adoption of the dual licensing model, there are no guarantees that such an approach will work for everyone. Stefan Landro tells a story about a company that changed their strategy away from the dual licensing approach: "I just recently spoke with someone that has developed a Ruby on Rails-like ⁶ framework for PHP, and they had tried out dual licensing for some time. They are changing strategy now, as they don't really make any money out of it. If people get the impression that something costs money, they won't use it. Their new strategy is to remove the commercial license and just use it because it is a helpful product. They also hope to attract new projects, and to derive more benefits from the community by having only the open source version. That's how it went with Middlegen; after a while outsiders contributed most".

Although they do not earn money directly from their open source efforts, Stefan Landro had some valuable input in relation with how an open source strategy can help in increasing the revenue. "There are several ways to earn money from open source. You can provide training, write books, help with installations, provide support, one might even get donations through Paypal. The market is very targeted toward knowledge, and providing certifications might also be an option. And of course, by initiating open source projects we are also profiling Bekk, making the company better known. This can help in attracting better employees. Visibility is important, and open source is a great way to achieve this".

7.2 Open Source is Not a Magical Bullet

While it might certainly be a smart decision to release the software as open source, it is important to be aware of the amount of work related to the adoption of such a strategy. One should always consider whether an open source strategy will indeed be helpful to the business, as a large amount of effort must normally be spent to achieve a successful project. Companies that seek much external participation must spend considerable time and money on community development, including

⁶Ruby on Rails is a framework for developing web applications

marketing to and recruiting potential contributors, integrating their efforts into the project, and developing a governance structure compatible with commercial and non-commercial constituents [WM05]. Thus, if the benefits of open source does not outweigh the costs associated with having such a project, it might be wiser to focus the efforts on other areas of the business. Thomas Malt describes the importance of earning money, and why they have been able to put more focus on Varnish due to receiving sponsoring to aid with the development: *“We have received financial support for the development of Varnish. Of course, this increases our motivation to develop and maintain the software, instead of selling other services. That is the problem, we have to earn money and then the incentives are much higher to use our developers for such purposes than to maintain old software”*.

Being a company that provide general open source consultancy services, he also has some experience with the work related to adopting such a strategy: *“I always emphasize this when I talk about open source to those that have their own projects, or are about to create one. Open source is not a magical bullet to get developers from all around the world to create the product for free. That is not how it works. Open source requires much work; You must have and maintain a good web site, each release must be buildable, runnable and contain documented improvements from the previous one. Once you start to release software that doesn't work as it should, you will lose both users and interested contributors. The same pertains to spending too much time between each release. Then people will lose their interest in the project. But at the same time, you must have something that works and is an improvement from the last time. If anything, open source projects demand more structure and more effort from the developers than closed source projects”*.

7.3 Summary

This chapter has described several of the aspects related to commercial adoption, and the rationale of all the studied subjects have been outlined. It has been shown that all these companies derive benefits from an open source strategy related with what was described in Chapter 6. Benefits are derived from the community, through increased company-internal knowledge, through sales of related services, as well as through licensing sales. In Chapter 6, possible business models were described. This chapter has also presented examples of the adoption of some of these business models, including; support seller, loss leader, the hybrid model, and the leveraging of the open source community. This chapter has also explained why it is important to consider the amount of work related to the release of open source software; It is not the magical bullet some people might believe. Releasing an open source project and to achieve success requires much efforts, and demands that a large amount of issues are taken care of. Chapter 8 describes some of the preparations that must be taken care of after the decision to go open source have been made.

Chapter 8

Important Measures to Achieve Success

This chapter goes through several measures that have been identified as highly important to achieve a successful project. Although the open source literature has not yet fully agreed upon what constitutes a successful open source project, it has been proposed that the success can be quantified by measuring the size of the development communities, the activity inside the projects, the time interval between bugs are reported to they are fixed, and the amount of downloads [CAH03]. Others have argued that to measure the success of a project, it should be evaluated by its alignment with the company's open source strategy, and that no measure matches all strategies [WM05]. While companies that try to accelerate the project's market share will be interested in the rate of the membership growth, and the rate of adoption, companies seeking cost reductions might be more interested in achieving a self-sustaining project, i.e., having a diverse collection of contributors and the project's ability to attract external contributions [WM05]. What is meant by success in the context of this chapter, relates to increasing the adoption of the project. The attraction of users and contributors are necessities for achieving success, independent on how one choose to define the term itself. This chapter describes how issues such as the project properties, the documentation, and the choice of an open source license relates to the adoption of an open source project.

8.1 Project Properties

There are several project properties that are of importance when releasing an open source project, and this section will describe the importance to have a good name for the project, code that works, a project meets a demand, and a well-designed architecture.

8.1.1 The Project Name

To have a good name that gives an idea of what the project does and that is easy to remember, might be helpful when releasing a project as open source [Fog05]. However, what is most important

related to the name, is that it does not infringe on any registered trademarks¹. As discussed in Chapter 5, if a trademark is infringed, it might be necessary to change the project's name to avoid lawsuits. This, in turn, might have ramifications such as having to make changes to the source code, as well as imposing a need to change a large amount of documentation, the text on the project web site, as well as switching the domain name, all of which will have costs associated. It might also have a negative effect to the brand of the software, if such has been built over time.

While it is highly important to make sure that the project name does not infringe on any registered trademarks, it might also be a good idea to perform some searches, and to choose a name that is not used for many other products. Certainly, no other open source project or computer software should use the same name as your project. This was noticed in relation with Keywatch, as several other products have the same name. At the time of this writing, a search on "Keywatch" will list the project web site, <http://www.keywatch.org>, ranked as number five on Google, and as number three on Yahoo. However, the first month, unrelated web sites ranked higher, and <http://www.keywatch.org> barely reached the top 100 on a direct search. Obviously, being ranked low on the search engines does not increase the adoption of the project². It is also beneficial that the domain name is similar to the project name, and that <http://projectname.com>, <http://projectname.org>, <http://projectname.net>, [country]³ are available. All these should be registered and forward the user to the main web site [Fog05].

8.1.2 Code that Works

There are examples of open source projects that have succeeded in attracting developers without having a project that works and can be tried out [vKSL03]. However, it has generally been agreed upon in the research literature that to have workable code is an important prerequisite for attracting contributors [Ray02][LT02]. Certainly, a project will not attract any end-users if the software does not work. When asked about what is the most important when releasing an open source project, Thomas Malt worded it in the following way: *"What is most important when starting an open source project, is to have a code that does something. It doesn't have to do much, but it must perform the basics. It must be downloadable, installable, and it must run without problems. Additionally, one must provide web pages that explain the product in an orderly, and elegant way"*. Thomas Malt went on explaining the release of Varnish, and the importance of providing some basic functionality: *"Varnish was developed as an open source project from the beginning. We had of course developed some functionality before the release. It is a requisite that a group of developers take on the responsibility to create some basic functionality before they release the software. Those that might have a need for the project, need to know that the software works when they download it and try it out. If it doesn't work at all, they will throw it away. The goal is that they think: "This is really cool and has much potential; if I just add this and this feature it will do what I need". That is how you attract developers."*

Eric Raymond has written something quite similar in the "Cathedral and the Bazaar" [Ray02]: *"Your program doesn't have to work particularly well. It can be crude, buggy, incomplete, and poorly documented. What it must not fail to do is (a) run, and (b) convince potential co-developers that it can be evolved into something really neat in the foreseeable future"*. Although he arguably

¹International trademark search is at the time of this writing available from the World International Property Organization(WIPO) at: <http://www.wipo.int/ipdl/en/search/madrid/search-struct.jsp>

²The importance of ranking high on search engines are further discussed in the next chapter.

³Such as .no, .se, .co.uk, etc.

understates the importance of having a well documented project as will be described in Section 8.2, he certainly emphasizes the importance of having a project that runs and has potential for success. The importance of having potential for success is presented in Section 8.1.4. Mozilla is an example of a project that had initial problems in getting external contributors [RdMF02]. It has been argued that these problems were much due to Mozilla's use of the proprietary Motif Library, and thus, potential contributors could not easily run and see the software working without having access to this library [Ray02].

The importance of having a project that runs can also be related to the argument in the literature stating that the users and developers of open source projects (at least initially), are often the same [GA04] [CLW⁺07]. It can as well be related to the meritocratic structuring that often takes place, as described in Chapter 4. Some users of an open source product will often become active users providing feedback and bug-reports on issues they identify in their use of the software, or they might even decide to help with the development efforts. To achieve this, the code must run. The following message from the Keywatch mailing list exemplifies how issues can be identified and how the community may provide a solution to the issue after running the software: *"In my use of Keywatch I mostly send a large number of events ,(could be 50 -5000) at the same time from my agent to the server. This calls for less use of the file system and I propose to be able to send many events in the same XML document. Proposed solution: Allow for more than one Event element within the Keywatch element[..] I have made a prototype implementation that includes my Java based agent and the FileHandler of the xmlprovider"*.

Thus, based on the literature, as well as the interviews, it can be said that having a project that works is a prerequisite to being able to attract users to a project, and a highly predisposing factor to the attraction of contributors. Potential contributors must have something they can try out and the software should work smoothly, or as Stefan Landro put it: *"The first impression is important. Everything must go smoothly when you download, install and run it"*.

8.1.3 Meet a Demand

To attract users and developers, it is also important that the software fulfills a demand in the community [BR06a][Stü06]. It might, however, be difficult to establish what the public demand is, just as it is with proprietary software [Stü06]. Potential competitors should always be studied to identify the distinguishing features of your project. If an existing open source project does the same as your own, and has an established position in the community, this will make it much more difficult to attract people to a newly started, similar project.

Keywatch is an interesting subject regarding the meeting of a demand. There are several high-profile systems that generally performs the same tasks, such as Zenoss⁴, and Nagios⁵, making it more difficult Keywatch to be widely adopted. Keymind has stated that the main advantages of Keywatch over its competitors is that it is based on the OSGi framework, providing high flexibility, and a plug-in architecture that makes it easy to create functionality directly focused one specific monitoring requirements. Since the project first was officially announced 2007-05-07, there have been a total of 145 downloads of the binaries⁶, and the web site has been visited by 1185 distinct IPs

⁴<http://www.zenoss.com>

⁵<http://www.nagios.org>

⁶There are unfortunately not any data available on connections to the source code repository

as of 2007-06-16. While not yet gaining widespread adoption, patience is important when releasing an open source project [Fog05], and it is as of yet too early to draw any final conclusions on the success of Keywatch. It is however difficult to say whether Keywatch does indeed fulfill a demand in the community. The lack of initial contributions and downloads might be due to the other projects mentioned fulfilling the demand for monitoring software.

The general notion is that the project must in some way provide something that can give a competitive advantage over existing products, i.e., the provision of more value to the users than what the existing projects provide. This might relate to being the only available open source product in a specific domain, or it might, e.g., include the provision of more and better functionality, better extensibility, or better usability. Generally, if the product is of higher quality than its potential competitors, or if it brings a degree of novelty, it will make for a potential successful project. Vidar Langseid attributed the success of eZ Publish to its technological grounding, and its customizability: *“We think that we have a better technological grounding than our competitors, and many of our customers have also noticed this. eZ Publish is very customizable compared to many other CMS”*. Qt on the other hand, was adopted into the KDE project and received a great amount of attention and adoption due to its position as the leading GUI toolkit for the Unix platform. Trolltech was able to achieve this even though much of the KDE-community at that time opposed their choice of license⁷.

Stefan Landro emphasized the importance of providing value, and of having a good product: *“If you have a really good product that is also perceived as such, then things can happen quickly. It is important that the project provides value outside of what is already out there, if not, then people aren’t really interested”*. He goes on attributing the success of Middlegen to how it fulfilled a need in the community, providing a tool that could save much time for developers in their daily work: *Middlegen was a project that intended to simplify everything in relation with middle-ware. It was a tool that generated the object model based on the database model. In many ways, Middlegen was an early such tool when there weren’t any good alternatives. It has been somewhat overrun now, by projects such as Spring and Hibernate. It was successful because it covered a need people had. It was a very good product back then and when people, e.g., know that there is a product freely available that can increase their productivity greatly, they will take an interest in it. Of course, there were many errors in it when it was released, however it is like that with all software”*.

Thomas Malt related the importance of meeting a demand to his thoughts on the success of Linux: *“What is most important is that the software meets the needs of those that might be interested to contribute. I am quite sure the first versions of the Linux kernel wasn’t optimally coded, but it met a demand of those interested in operating systems, thus they contributed anyway”*. While Linux is a great example of a product meeting a demand as was also described in Chapter 3, Mozilla Firefox can also be brought forth as an example since it was a superior alternative to the Microsoft Internet Explorer when it was released [Kri05]. It provided better compatibility and adherence of standards, tabbing, and better security. Thus, Mozilla Firefox was in accordance with these words from Stefan Landro: *“What is important is that you have a product that is of high quality. It shouldn’t do too much, but rather solve a new, easy problem. What the software does, it should do well”*.

⁷The reasoning behind KDEs adoption of Qt is described at: <http://www.kde.org/whatiskde/qt.php>

8.1.4 Potential for Success

It is important that the project shows future promise, as this might make it easier to attract developers. Early contributions will often be very visible in a successful project [LT02], and this can act as a motivating factor, i.e., related to future monetary rewards as described in Chapter 5. To show potential for success, the project should be of high quality, run, meet a demand, but at the same time provide interesting tasks that need further work. The program itself must appeal to potential contributors. Christian Schwartz provided an example of how projects that have potential for success can increase the motivations to contribute: *“Many perceive open open source as a way to build their career. If you are the main contributor in a successful project, you become a much more attractive prospect for future employers”*. It is important not only that the project shows future promise, but also that external contributors can identify sections where they can uniquely contribute. If they do not see how they can “make their mark”, they might be less inclined to contribute to a sponsored project [WM05].

It is, however, important to be aware that people are different, and what one person finds appealing, others might not. To attract people that take on a variety of roles in a project, can help it tremendously as explained by Thomas Malt: *“You will find that in some successful open source projects. They were perhaps not perfectly coded originally, but it has attracted users and developers that have improved it incrementally over time. People have cleaned up the code. There are many different people; Some are very creative and come up with good ideas, create the project and develop the initial runnable release. Others see the potential in this code and are good at creating structure-to clean up. Finally, you have those that do the polishing, they improve the documentation and clean up the code even more. The right combination of people is important in all software development projects, not just in open source”*. Companies should be aware though, that company-internal employees might have to take on certain roles, especially related to the provision of documentation.

The timing of the release might also prove essential, as this can influence the potential for success. Vidar Langseid emphasizes this subject in relation with eZ Publish: *“I think the timing was very important for our success. We probably couldn’t have followed the same actions today with the same amount of success[.] That we went open source as early as we did, and at the same time targeted the international community were of great importance in achieving international success”*.

8.1.5 Software Architecture

“The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements and the relationships among them” [BCK05].

Christian Schwartz explains from his experiences as an open source developer, how difficult it can be to understand the code of others, and relates this to the software architecture: *“To understand the code written by others can be difficult, especially if it has been put together ad-hoc by many different people. This is especially true for certain old projects that have an outdated architecture, and that have been extended too much with new features. Projects that wasn’t made to be as extensive as they have become. This is a reason as to why new projects emerge that perform similar tasks as the older ones do, only better”*. To increase the chances of making the process of releasing code as open source successful, it is important that the existing architecture is suitable for attracting

contributors, i.e., having a modular architecture, or as Tomas Malt put it: “[..] *It is very important to have a clean and structured code base*”.

Modularity

It is of huge importance to have a modular architecture, in which each module in the software has well-defined areas of functionality with clearly defined interfaces. Modularity increases the flexibility and comprehensibility of a system allowing a reduction in development time. While software development in general benefit greatly from such an architecture, it can be viewed as an absolute necessity in open source projects. Indeed, it has been stated in the literature that if the software is not modular, it can generally be said that it is not suitable for open source development [KM01].

Software featuring high modularity makes distributed functionality possible, allowing for extending the functionality of the product without changing what already has been created. From a pragmatic viewpoint, this allows a company to maintain control of the original product, while allowing the functionality to extend and scale with the community [O’R99].

Modularity is also important for attracting newcomers by allowing a gradual learning curve. A modular architecture causes increased transparency, which makes it easier to take advantage of specialized knowledge [vKSL03], and it lowers the entrance barrier increasing the chance of attracting new contributors [Stü06]. Additionally, it lowers the domain knowledge that is required for participating in the project, i.e., it will not be necessary to have full understanding of the entire architecture to provide beneficial contributions. A greater number of modules can offer more opportunities for recognition, which can further enhance contributors’ motivations [BC03]. This is especially important related to open source development. Outside contributors can choose from a wide variety of projects, and will typically contribute in an area which they consider themselves knowledgeable. Thereby, an expert in a certain area can provide contributions in relation to his area of expertise. Bekk can be drawn forth as an example in that they focus on writing clean code with emphasis on modularity: *“We are very careful to write clean code with emphasis on modularity. We refactor all the time, and are very aware that someone else will read the code at some point. You want to hold a certain standard when you know that others can see what you have done. You don’t want your name on any garbage”*.

The modularity of the architecture should also accommodate for the development of plug-ins and add-ons to the product, which both are popular within the community and value-adding for the product [Sca02]; Thomas Malt: *“Software that have plug-in architectures such as Eclipse and typical CMS’, need to handle plug-ins and modules in a proper way. The better the architecture, the easier it is to create, and integrate new modules. Everything become much more agile”*. eZ Publish is an example of a CMS that provide a modular architecture to encourage the development of plug-ins, also when it was initially released: *“The project had quite a modular architecture when it was released. That was an important goal all along. We went through typical analysis and design phases having a professional development process. This was something we got credit for in the beginning. That the code was clean and easily extensible, made it easy to create plug-ins. But of course, we have several times experienced that the modularity wasn’t quite as good as we had hoped for when implementing new features”*.

Another benefit associated with providing a modular architecture, is that the loose coupling associated with it makes the software more maintainable, leading to reduced ripple effects caused by maintenance work. Hence, the conclusion is that if modularity is not present in the current architecture, the necessary modifications must be made to make it so, although a careful consideration of such costs should be made. The developers must be willing to clean up, refactor the code, and provide a beneficial architecture. The code should be easy to understand for outsiders.

Adherence to Open Standards and Protocols

One of the important requirements to sustain innovation, is to adhere to open standards and protocols [O'R99]. Open source software is well known to have a high rate of adherence to open standards. This allows for easy connection between new and existing software, as well as easier porting across platforms. Firstly, by adhering to open standards, the software is easier accepted by the open source community. Secondly, the simpler porting and combining of software solutions can cause an increased user base as well as the development of coexisting value adding software [ES06].

Software Components

There are also certain considerations related to software components that must be made when releasing software as open source:

Commercial Components

COTS⁸ components that are used in the system can make it impossible to release the software as open source. These components will often include a non-disclosure clause stating that redistribution of the source code and/or the interfaces is not allowed. The possibility of finding another suitable component to use is of course an alternative that can be considered in these circumstances, although intensive cost-benefit evaluation may be necessary.

Open Source Components

If the software has previously been sold without violating any of the components' licenses, e.g., only comprising BSD licensed software, then distribution of the entire product as open source should not cause any problems from a legal point of view.

If the software has been used for internal purposes only and is based on open source components, then the licenses' intercompatibility might become an issue. This is because open source licenses come into effect only when the software is distributed in any way, as described in Chapter 4. If the software comprises several open source components, then the licensing scheme of each and every component must be assessed considering compatibility. Especially GPL has a large amount of open source licenses of which it cannot be combined with⁹. Additionally, open source software is often a compilation of code from many different sources, making it hard to detect whether parts are protected by intellectual property rights. Therefore, an assessment of the components must be taken with regard to property rights. The legal exposure should be minimal if a component has a strong open source community, is mature, is widely adopted, and has a clear legal status [RE04].

⁸Commercial Off-The-Shelf

⁹The licenses of which GPL is incompatible with can be found at: <http://www.gnu.org/philosophy/license-list.html>

8.2 Software Documentation

The common denominator that helps people to install, run, use and contribute to an open source project, is the documentation. Software documentation relates to program listings and technical manuals that describe the operation and use of the software¹⁰. Thomas Malt explained the importance of providing good documentation: *“I am very focused on the documentation. The API documentation must be 100 percent, each and every function should be commented. To attract contributors, it is important that the code is understandable and possible to work with. It is necessary to provide a well-written read-me, a good install-script, and to have the same information available at the web page. FAQs are great, but they aren’t useful if they haven’t evolved naturally. People see through FAQs that don’t consist of true questions”*.

8.2.1 User Documentation

User documentation consists of all documents intended for helping the end-users of the software, and comprises information on how to install and use the program. Stefan Landro describes the importance of providing good user documentation in relation with open source projects: *“You need documentation so that people know how to use the software. It is no use if it looks cool, but people don’t understand it. It is important to have things such as getting-started guides, so that people can see something on the screen, become impressed in a short time. Providing videos showing the use of the product, quick-start, and step-to-step guides with some screen-shots etc., can help a lot”*. When people abandon, they abandon early; therefore, it is the earliest stages, like installation, that need the most support [Fog05].

Generally, several variants of the documentation should be provided simultaneously, as each document has a special purpose in aiding the users. The provision of good enough user documentation is often a challenge in open source projects [LT02]. The developers of an open source project might find it difficult to identify what might cause outsiders problems in the use of the software, and they might also lack the motivation to write documentation that explains tacit knowledge. User documentation should be organized properly, so that the users can find what they are looking for, i.e., to avoid that they throw away the software or submit invalid bug reports due to lack of knowledge.

Even though the user documentation is very important, there are many examples of open source projects that have been successful without it. eZ Publish is an example of a project that did not provide much user documentation upon release. Vidar Langseid explained this issue in the following way: *“There was very little user documentation when we released eZ Publish. But then again, the project wasn’t as big at that time as it is now. There were of course information on how to install, that is extremely important. However, pure how-to-use documentation was lacking. We did have some technical documentation, but the feedback was that also this could be improved. Also, I don’t think we had the technical documentation available at the web page, but we put it there after a while”*. While he stated that the user documentation was not as good as it should have been, he still emphasized what is of major importance; that there is enough documentation for people to understand how to install and run the software.

¹⁰Paraphrased from Wordnet: <http://wordnet.princeton.edu/>

8.2.2 Development Documentation

To receive contributions from the community, it is extremely important that they are able to understand the code and the architecture without having to scrutinize the source code, and the provision of development documentation such as of the API, the architecture, and the design can greatly increase the possibility of attracting contributions. As the external community in open source projects are introduced to the project after the initial design and architecture have already been created, the lack of proper (development) documentation can provide a significant entry-barrier to newcomers, making it harder to gain an overall understanding of the underlying architecture [vKSL03]. The provision of proper development documentation can also help increasing the maintainability and quality of the source code, if everyone adheres to the same coding standards outlined in it [Mic05].

For the GUI toolkit Qt, providing enough developer documentation is paramount for it to be used by others: *“Our documentation is among the best available, and I don’t know of any project that has better documentation for the developers. We have developed all the documentation ourselves, and we have several employees working on it. The documentation is one of our major sale arguments”*.

We saw in the introduction to this section, that Thomas Malt emphasized the importance of providing a fully documented API. To ensure that this is achieved, tools such as Javadoc can be used. Javadoc is used to generate API documentation in HTML format. It is freely supplied along with the Java SDK, and is widely used in both open and proprietary development efforts. It works by automatically extracting information about package structures, classes, method signatures and more [Les02]. Its benefits can be greatly improved when the developers include some explanatory comments about the properties that are to be extracted.

8.3 Choice of License

A quite thorough description of open source licenses was provided in Chapter 4. The choice of licensing scheme might have an impact on the community; who is attracted to it as well as the end-users of the software. It might also influence other projects that compete with or complement your product, as well as commercial vendors and support providers [LT05b]. There is a large amount of different open source licenses, and new ones are often written pertaining to specific projects. When choosing a license, it is in most cases wise to use one that is well known. These have typically been written by lawyers, and some of them have already been tried in the court. Additionally, as open source developers mostly will be somewhat familiar with the most common licenses, they will not be drawn away from projects due to not understanding what some unknown license might entail. Indeed, OSI has initiated a proliferation campaign to encourage the use of the most common licenses. They think the others serve little purpose other than to steal attention from developers evaluating their choice of license for some new project ¹¹.

¹¹Information about the OSI proliferation campaign, and their suggested licenses can be found at <http://www.opensource.org/proliferation-report> (All of the licenses mentioned herein, and in Chapter 4 are encouraged by OSI)

8.3.1 Competitors and Complementary solutions

If the company is being out-competed, then licensing the source code under GPL might be a possibility to try to steal away market share from the competitors. The other licenses mentioned in Chapter 4 are not very viable solutions here, taking into consideration that the software then might be exploited by the competitors.

If the competing open source software is using less restricting licenses such as the BSD, then your own software should probably also stick to the same license. This is due to complementary strategies and network externality effects [LT05b]. Generally, if a certain license is used by most related projects and components, the same license should be adopted.

Target Users

The target audience can have an impact on the license choice. If the target audience mainly comprises users of a UNIX based operating systems, then GPL is probably the choice that will attract most contributions. If the target audience comprises non-technical users and the software is mainly intended for end-use, then the choice is more open. Especially related to end-use for Windows platform, GPL is probably not the best choice [Kri03]. Another license should be used instead. If the target is developers that want to use the software commercially, i.e., for the creation of plug-ins, then GPL is unsuitable, and another license such as the EPL (specifically designed for this purpose), should be used.

GPL and LGPL

If GPL components are incorporated in the software, the rest of the project also have to be GPL-licensed. Hence, GPL can be chosen to show the community good intentions as of why the source code is released, or when the source code is not to be used by other commercial actors in their projects. It helps protecting the work of the community from commercial hijacking; however only value-adding strategies are viable with code licensed with GPL¹². Use of GPL can be a good choice if the goal is to accelerate innovation and attracting contributors, while not wanting commercial actors to use the source code in their software. GPL is especially well-acknowledged within the POSIX¹³ community, and software developed exclusively for such platforms might benefit for the use of GPL. Generally, if the negative effects that GPL imposes on the company are not impacting the company's other strategies in a negative way, then GPL is the best choice. GPL should also be the license of choice together with a proprietary license if a dual licensing strategy is adopted. By choosing GPL in this case, the community will be able to use the code and receive the benefits that GPL provides, while the company at the same time can charge commercial actors for use of proprietary licensed code, as companies cannot use the GPLed version. Such a strategy will require full ownership over all of the source code, and will make it impossible to use the community's contributions in the software that is sold commercially¹⁴.

Use of the LGPL is recommended if the program is a library as it is both compatible with GPL and can be combined with proprietary works. As the licensed code otherwise works as GPL, it can, e.g., help promoting standardization of a library. Of course, adherence to the licensing terms of

¹²Unless you have copyright over the entire code base, as this enables the possibility of dual licensing.

¹³Collection of standards that define the API for software compatibility with UNIX

¹⁴An entire rewrite of the submitted code might be a possibility as copyright only protects the expression and not ideas

LGPL will in such cases be necessary, meaning that eventual plugged-in proprietary code must be properly connected.

BSD License and the Apache License

The BSD License and the Apache License are known as being permissive, and should be used in cases where this will help the project, such as to encourage commercial users to adopt the project. These licenses are suitable for companies adopting a loss leader strategy. Stefan Landro argued in the interviews for the use of these licenses: *“It is important to have a sensible license. Most successful projects use a BSD-style license¹⁵, such as the Apache License. It is important that the license is well known, and has a good reputation in the community”*.

It should be noted that the Apache License cannot be combined with GPL¹⁶, making (Revised) BSD license a possible alternative in the case that commercial use is intended while still trying to leverage the GPL-community that might want to use your software. If trying to establish a standard, then the use of an unrestrictive license such as BSD or the Apache license makes sense. If the project has strong appeal and contributors might benefit considerably from signaling incentives, or the licensors are well trusted, then the use of a BSD-style license might also be a good choice [LT05b]. The main negative point with the permissive property of these license is the risk of project hijacking.

Mozilla Public License and the Eclipse Public License

These licenses can be referred to as being weakly copyleft¹⁷, and encourages open competition among commercial and non-commercial developers [Hec00]. These licenses are not compatible with the GPL, hence use of these licenses means that the benefits such compatibility can provide will be neglected. They should probably not be the license of choice if the target operating system of the software is POSIX-based. They do, however, encourage contributions back to the community in a larger extent than the BSD-style licenses.

8.4 Summary

This chapter has explained several measures that are important to increase the chance of an open source project being adopted, thus being important to enable a successful open source project. It has explained the importance of providing software that works and can be tried out, the importance of having software that meets a demand in the community, as well as why issues such as showing promise for future success, and having a proper architecture are important. The chapter has also given an explanation of why good documentation can help in attracting users and contributors, as well as describing the importance of choosing a good license for the project. However, these issues are only the preliminary requisites to achieve a successful project; Hosting, presentation and old-fashioned marketing are indispensable. Considerations related to these are described in the next chapter.

¹⁵Licenses that are similar to BSD License and the Apache License, i.e., the source code licensed under such a license can be used for any purpose. MIT License is an example of another such license, being essentially similar to the revised BSD.

¹⁶This is changed in the upcoming GPL v3

¹⁷See chapter 4

Chapter 9

Releasing the Project

This chapter addresses research question 2, and relates to the actual release of an open source project. There are several issues that must be addressed in the release process; Infrastructural decisions must be made, a web page that enables collaboration must be created, and the project must be announced to attract contributors. This chapter discusses the release process based on interviews, direct participation in the release of an open source project, and the evaluation of the project sites of several commercial open source initiatives.

9.1 Tools Enabling Distributed Collaboration

In an open source project, the community of users and developers is distributed across the world, and to have a proper platform for collaboration and communication is vital to the project's success. The use of tools that makes it possible to communicate, both in an asynchronous way and in real time, makes it possible for the community to ask questions, exchange ideas, and to coordinate development efforts. Tools enabling software configuration management (SCM)¹ are imperative to receive to developing software in a distributed environment, and to create an awareness of the development process. In addition to looking at such tools, this section will present the concept of wikis, and how these can benefit the creation and maintenance of documentation in open source projects.

9.1.1 Mailing Lists

Mailing lists are often regarded as the most important tool for communication in open source projects [Fog05]. Mailing lists allow members to send email messages to all group members [BSKK02], and are “[..] *very effective to coordinate development efforts*”, as stated by Thomas Malt. Typically, all important messages will be posted to the mailing list, and the broadcasting of all communication makes the development work transparent, providing awareness in the community of what others are doing [YYSI00][DM05]. Access to the mailing list should be provided through email and web-based subscription, together with some software for managing the mailing

¹SCM is concerned with controlling changes to software products [CW98].

list such as Mailman². Such software enables possibilities for specifying the settings of the mailing list, and is invaluable to prevent spam. It is also imperative that the messages in the mailing list are archived and that these archives are searchable, as this ensures that newcomers and users have access to the previous postings

9.1.2 Forums

Forums are web applications that are commonly used as a medium for communication in open source projects. They allow anyone to post to a designated group location where others can read and comment on those messages [BSKK02]. It is generally necessary to register and log onto the forum to be able to post messages, and forums might not be as an effective tool for coordination as mailing lists. At the same time, they are less invasive - it is required to actively seek the forum to view the posts³. Christian Schwartz argued in the favor of the use of forums: *“It is not as scary for users to post at forums. You don’t end up spamming with unimportant requests such is the case with mailing lists, since the choice of what to read is voluntary when forums are used”*. Forums incorporate automatic archiving of all posts and are generally searchable, meaning that no specific actions must be taken to ensure the availability of old postings.

9.1.3 Internet Relay Chat

IRC is a real time chat service that can be used by open source projects to accommodate for quick, lightweight discussions as well as as an aid to establish a larger sense of belonging to a community. IRC can be an effective means of communication in open source projects, but some persistent media as mailing lists or forums should be used for important discussions. When using IRC, the channel name should be the same as the name of the project and information on how to connect to the right server and channel should be easily available from the project web site [Fog05]. The communication in IRC channels can be archived. Nevertheless, discussions that are subject to archiving should take place in other media.

9.1.4 Instant Messengers(IM)

Instant messengers, such as MSN, Skype, AIM, and ICQ can be used much in the same way as IRC. The difference here is that these media are designed for one-to-one communications, and thus they do not establish the same sense of belonging to a community as IRC might provide. Vidar Langseid described their use of IM in the following way: *“We don’t use IRC today. But we have used it for communication earlier. Today, it is mostly through instant messaging and email. We use instant messaging internally and toward customers. The sales department uses Skype quite a lot”*.

²Mailman is used in the Keywatch project.

³Unless modifying the settings so that posts are received in the email

9.1.5 Revision Control Systems

Revision control systems⁴ are applications designed to synchronize work and to keep track of changes in the source [vKSL03]. There are several different revision control systems, and two of the most commonly used are Concurrent Versioning System(CVS) and Subversion(SVN). CVS was developed in 1986, based on the already existing Revision Control System(RCS), and SVN was created in 2000, intended to act as a replacement for CVS. Subversion generally provides the same features and interfaces as CVS, but fixes some of the problems identified with the older software⁵.

Revision control systems are vital for any open source project, since they enable developers to manage code changes made in distributed environments in a convenient way. Further, they enable everyone to check what is happening to the code base. Revision control systems help with virtually every aspect of running a project: communication, release management, bug management, code stability, and experimental development efforts [Fog05]. Files, changes, and user comments are stored in a central repository, which is kept separate from users' local versions. People can download the content of the repository, making it possible for developers to work on the same set of files as shown in Figure 9.1. The changed files can then be added back to the central repository, and in most cases these systems will merge conflicting changes, i.e., when two users have revised the same document. The history of a project will be stored by the server, making it possible to compare the changes that have been made, and to rollback to an older version if necessary.

In open source projects, there are generally only a few trusted developers that have commit-access to the code repository⁶, however everyone are allowed to view and download the code. This makes it possible to contribute even though changes cannot be made directly. Every commit should generate an automatic email or post showing what changes have been made and by whom, and published to a designated mailing list or forum used for such purposes [Fog05]. This process encourages peer review, but also helps to create a sense of belonging to a community. This is because the messages establish a shared environment in which people can react to events (commits) that they know are visible to others as well.

9.1.6 Bug Tracking Systems

Bug Tracking Systems⁷ are applications that can keep track of bugs and other issues that relate to the software [Fog05][SC05]. They provide several features for software configuration management; they can help with coordination, the planning of releases, and generally to keep track of anything that has distinct beginning and end states [Fog05]. There are several such systems, both as proprietary solutions and as open source [SC05]. The most commonly used bug tracker software is Bugzilla, of which the issue states are shown in Figure 9.2 [ES06]. Bugzilla is a web application, and users only have to interact with its web page [SC05].

Anyone may file an issue, anyone may look at an issue, and anyone may browse the list of currently open issues. It is important that the bug tracker is linked to the revision control system, and the persisting communication tool, to keep everyone updated about the status of the reported issues.

⁴Also known as version control and source control systems

⁵A listing of the changes from CVS can be found at: <http://subversion.tigris.org/>.

⁶Allowed by the project administrators to commit changes, adding, deleting, or modifying files.

⁷These are also referred to as Issue Trackers

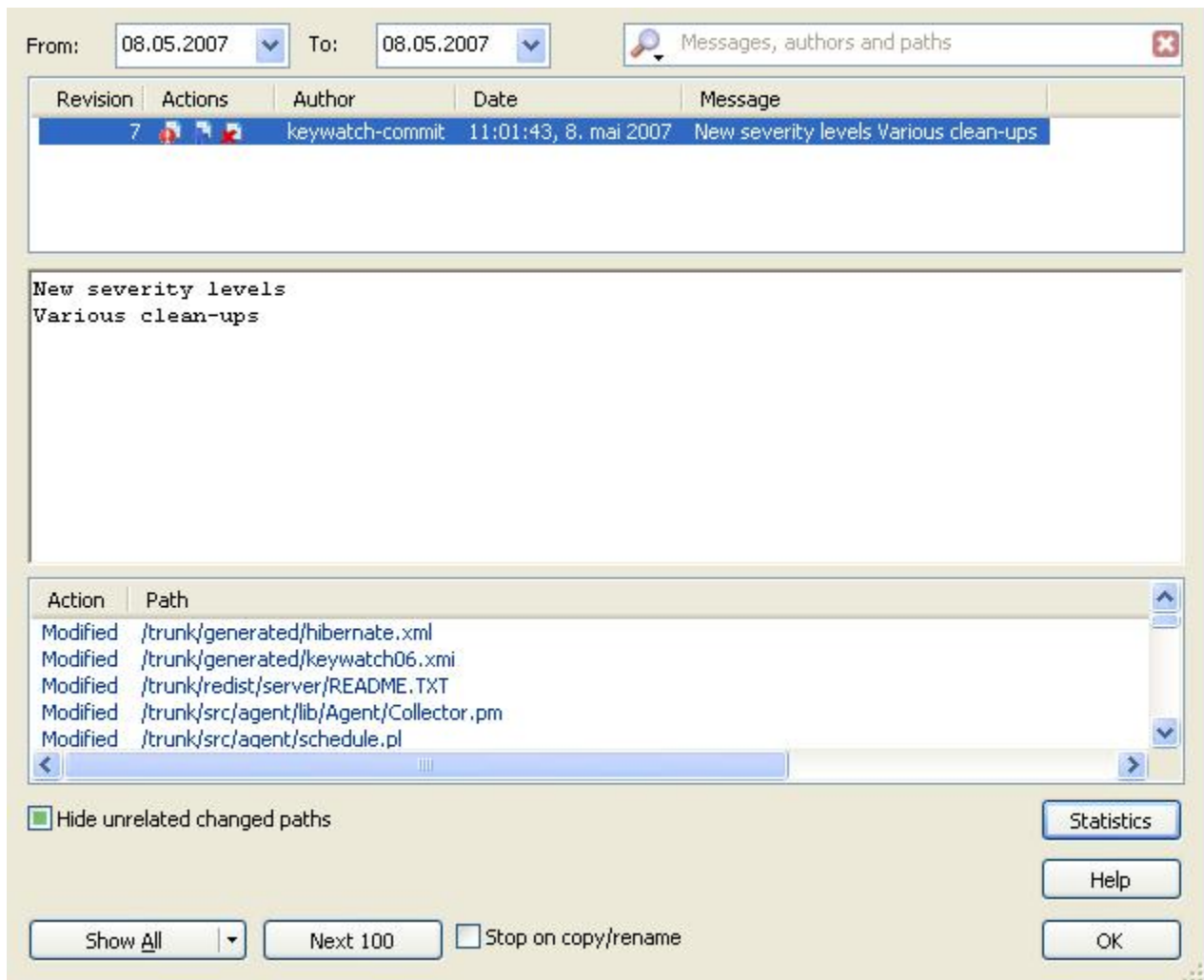


Figure 9.1: Using SVN with Keywatch

Even though the bug tracker should allow for anonymous reporting, they should encourage the provision of contact information. Thus, the person that have reported an issue can be contacted for more information [Fog05].

9.1.7 Wikis

Wikis have lately emerged as a new commonly used tool for writing documentation. They were invented in 1995 [LC06], but has not gained much popularity until recently, partly boosted by the success of the online encyclopedia, Wikipedia⁸. Wikis are implemented as a website-component, a CGI script or similar technology [Aro02] and provides an easy interface for writing documentation in a collaborative effort. This relates both to technical documentation as well as documentation

⁸<http://www.wikipedia.com>

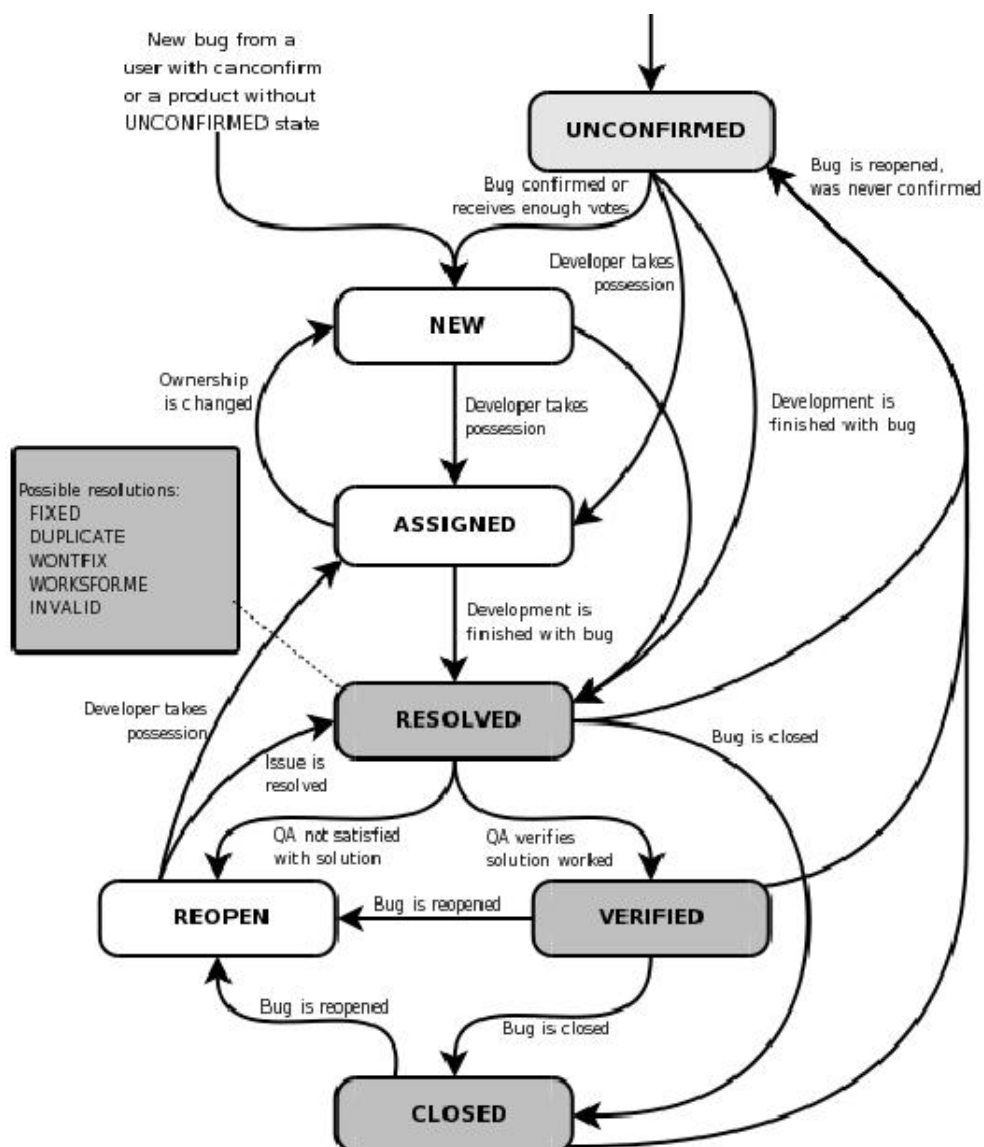


Figure 9.2: The issue states in Bugzilla

for end-users. Visitors can access the information in the wiki as regular web sites, and can then view the existing content and, depending on the access rights that are set; modify, add and delete information from the Wiki. The major advantage of wikis over the editing of regular web pages, is that making changes to the content involves less interface overhead than the editing of regular web pages.

eZ Systems has experienced the value of having a Wiki as an aid in their open source projects: *“We used a system much like wikis where everyone could add content. We didn’t really get much contributions to this system and thus created most of the content ourselves. There were although some people that posted errors they had found. We went away from that solution a couple of years ago, and the community created their own wiki shortly after. This wiki is technically focused,*

and we have our own documentation that supplements it. We have began to link together our own documentation and this wiki, so its just recently it has evolved to a significant size". What Vidar Langseid described here, exemplifies a concept related to the use of wikis; A wiki will not automatically enforce a large amount of visitors to add high quality content. Thomas Malt was also positive to the use of wikis, mentioning the importance of good structuring: "*Wiki is a great medium for documentation if it is correctly structured and you receive some constructive contributions*".

Wikis come with some limitations such as lack of navigational principles and possible duplicates of information. It is important to have editorial standards, and to show how the wiki is to be used by editing all pages to adhere to these standards, i.e., structure the wiki in a proper way, as mentioned by Thomas Malt.

9.1.8 Choice of Tools

The choice of which tools to provide is an essential part of the release process of open source software. In general, it can be said that all of the tools mentioned above can coexist and might help to enable communication and to attract developers. Thomas Malt explained his experiences of what tools are important in open source projects: "*My experience is that forums are taking up a larger amount of the communication these days. For active developers, the mailing list and the Wiki are most important. Mailing lists are very effective to coordinate the development efforts, while forums are more used by users that don't know the project very well. The Wiki is a very effective tool for documentation, design suggestions, and so on. In addition, all the developers use IRC a lot. They discuss possible solutions back and forth in dedicated IRC-channels*".

The available tools at the surveyed web sites can be seen in Table 9.1. All provided a tool for communication, and most also used a bug tracker. Interestingly, only 50 percent of the examined project web sites presented public source code repositories accessible through revision control systems, something that may warrant further studies.

As a minimum, a mailing lists or a forum must be available to allow communication. It is also highly recommended that a revision control system for the source code is provided upon the initial release to enhance the collaborative development efforts. It is not necessary to provide all of the other tools at once, as the amount of activity in the beginning probably will be limited. This approach was taken with Keywatch, and indeed, there were not much activity the first month it was available (as can be seen in figure 9.3), suggesting that providing a forum in the near future is not very important.

It is natural that the infrastructure is modified over time, as have also been the case with Linpro's open source initiatives: "*The infrastructure has been modified over time. We have gone from using regular mailing lists and static web pages to using bug trackers and wiki in addition to the mailing lists and forums*". Indeed, providing more tools for communication than what is necessary might be counterproductive [Fog05], and the provision of more means of communication, e.g., the creation of new mailing lists and forum categories should be performed when it becomes natural. The initial choice then, relates to whether to choose a mailing list or a forum when the project is released.

"*I don't think it really matter whether a forum or mailing lists are used for communication, as long as one of them is present*", Stefan Landro stated when asked about his thoughts on the subject. However, it is suggested in other interviews that what to use depends on the project and the target

Project	Available Tools
netWiser	Issue Tracker Mailing Lists: User and Developer No Revision Control
Openbravo	Bug Tracker - available from SourceForge Subversion - available from SourceForge Wiki Forum: Several Categories - available from SourceForge
OpenEMM	Bug Tracker - available from SourceForge Forum: Several Categories Wiki No Revision Control
TYPOLight	Forum: Several Categories Wiki No Revision Control
Visual WebGui	Bug Tracker Forum: Several Categories No Revision Control
Webload	Forum: Several Categories Subversion - available from SourceForge Wiki

Table 9.1: Tools available at surveyed web sites

users. Thomas Malt put their experience with these tools the following way: “*We don’t have any separation such as having mailing lists only for the developers or forums only for the users. But we have noticed that many new people, those that haven’t grown up with text-based mailing lists and news groups, prefer using forums*”. Vidar Langseid and eZ Systems have also noticed a similar phenomenon, although from a different perspective: “*We have had some discussions on whether we should use mailing lists or forums for communication. That has been repeated several times. We have now ended up with providing only mailing lists for eZ Components, as this is targeted toward developers. We did provide a forum initially, but it wasn’t much used. We only had one mailing list for eZ Publish in the beginning, but now we use both forum and mailing lists. It varies where the discussions arise. What seems typical is that use-related discussions take place at the forums, while the more technical discussions arise on the mailing lists. We will continue to use both for eZ Publish*”. If the project targets people that might not be common users of mailing lists, then a forum might be a good option. If it mainly targets developers, then a mailing list should be the natural choice, and a forum is initially not necessary.

Trolltech did not use a bug tracker initially, but they did implement one after a while: “*We communicated through news and mailing lists. We also had our own homepage, but no bugtracker. We got the bugtracker around Christmas 1997. We also used CVS, which was extremely important*”. The bug tracker should, if possible, be available when the project is initially released. However, if one implements these solutions manually, the integration of all the tools might take some time.

May 2007 Archives by thread

- ◆ **Messages sorted by:** [\[subject \]](#) [\[author \]](#) [\[date \]](#)
- ◆ [More info on this list...](#)

Starting: *Thu May 10 14:38:48 PDT 2007*

Ending: *Mon May 21 23:49:23 PDT 2007*

Messages: 8

- ◆ [\[Keywatch-dev\] Help getting started](#) *Helander, Lars-Erik*
- ◆ [\[Keywatch-dev\] Help getting started](#) *Stein Roger Skafløtten*
 - [\[Keywatch-dev\] Help getting started](#) *Helander, Lars-Erik*
 - [\[Keywatch-dev\] Help getting started](#) *Stein Roger Skafløtten*
 - [\[Keywatch-dev\] Help getting started](#) *Helander, Lars-Erik*
- ◆ [\[Keywatch-dev\] Event XML modifications](#) *Helander, Lars-Erik*
- ◆ [\[Keywatch-dev\] Event XML modifications](#) *Helander, Lars-Erik*
 - [\[Keywatch-dev\] Event XML modifications](#) *Stein Roger Skafløtten*

Last message date: *Mon May 21 23:49:23 PDT 2007*

Archived on: *Tue May 22 00:53:19 PDT 2007*

-
- ◆ **Messages sorted by:** [\[subject \]](#) [\[author \]](#) [\[date \]](#)
 - ◆ [More info on this list...](#)

This archive was generated by Pipermail 0.09 (Mailman edition).

Figure 9.3: Communication in the Keywatch mailing list, May 2007

Regarding the importance of wikis; while being a great tool for use with documentation, what is of major importance is that the documentation is available.

9.2 The Web Site

The project web site is the project's external interface to the community, and it is very important that it presents the project in a beneficial way. The web site layout- and design should enable good usability and a good general user experience. At the same time, the content of the web site must include all important general information, documentation and links to the tools that are used. However, as the web site is the entry portal for both regular visitors and developers, the principle of scaled presentation should be adhered, meaning that what the degree of detail presented should correspond to the efforts put in by the reader [Fog05]. Unless the web site meets the needs of the intended users, it does not meet the need of the organization providing the web site [Bev04].

9.2.1 Layout and Design

The very first thing a visitor learns about a project is from what its web site looks like. This information is absorbed before any of the actual content is comprehended, before any of the text has been read or links clicked on [Fog05]. The design and layout of a web site relate to its visual presentation by the means of a background color, white space, font size and color, and other design elements, and directly affects the usability of the web pages [BM01]. The layout and design of the web site can be an important aid in creating interest for the project. *“I think having a good web site is an absolute necessity”*, Thomas Malt stated when asked about the importance of the web site for open source projects. Also Vidar Langseid emphasized the importance of having a good web page: *“I believe that the design and layout of the web site are important to create an interest, however we haven’t been particularly focused on these issues, at least not until recently. Since our competitors have put much effort into the layout and design, we also have been required to consider this part. The first impression is important, and it is more important these days than before. Earlier, it was mostly technical people that used our software, and they didn’t care as much about the visuals. However, we are a totally different sales organization today than we were back then. Now we are also marketing our software actively, trying to sell the software”*. A professional look of the web page can subconsciously cause the viewers to think more highly of the project, to encourage them to trying it out [Fog05]. Knut Yrvin related the provision of an improved web site to increased adoption and sales: *“Our web site has been heavily improved, as it clearly is our most important channel for selling software. We are still working at improving the usability, but it is only small adjustments. After the overhaul of the web page, our sales have increased. The web site is vital, it is through the web site we get our customers”*. However, to create a professional web site costs a lot of money, as Thomas Malt described: *“We have of course provided a simple and functional web page, but we haven’t spent time to create the most easily marketable web page. I believe this is a weakness, but it is a result of cost and benefit. It costs a lot of money”*.

An initial web site layout and design was created for Keywatch. They had set an initial requirement that it should be simple, clean, and easy to use. The process used to create this web page were as following:

- Modeling of the navigational structure
- Sketching of the design and layout
- Creation of the structural layer (XHTML markup)
- Provision of mockup content
- Creation of presentational layer (CSS)
- Testing of the web page

To create a web page that provides good usability, it is important to decide upon how it is to be navigated. User’s expectations should be met by following conventions established by other sites, users should know where they are and to where they can go, and links should be self-explanatory [Bev04]. Thus, the navigational structure was inspired by the web sites of several other open source projects, the menu items were given self-explanatory names, and it was decided to use

“breadcrumbs”⁹, to help users identify where they are at the page. The content that Keymind had decided to provide with the initial release emerged as natural menu items as seen in Figure 9.4.

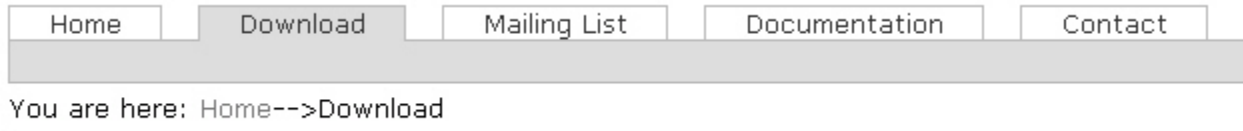


Figure 9.4: Menu structure of the Keywatch web site

It was decided to use a horizontal menu, put at the top of the page. The initial thought was to use a three-column approach with the main navigational menu to the left. Also other approaches were considered, and three different layouts were presented to Keymind. It was decided to adopt a two-column approach. The navigation menu was decided to be put at the top of the page with the main content below the menu, and with the column to the right set aside for login, search, RSS feeds and links to related sites.

It was decided to ensure good separation between document structure and presentation. Thus, the structural layer of the web site was created using Strict XHTML 1.0¹⁰ as structural and semantic markup, leaving all the presentation to be handled by Cascading Style Sheets (CSS). CSS is a W3C¹¹ standard for defining the presentation of web documents [Rob06]. The web site was logically divided into sections using the “div” tag, and the elements that were to be presented in a specific way were also given “class” and “id” attributes, to be accessible by the “class” and “id” selectors in the stylesheet, as well as to be accessible through the Document Object Model for future scripting purposes. Some mockup content were added using a “lorem ipsum”-generator¹², to ensure that the web page would look as it should when the final content was provided, and “Breadcrumbs” were added for better usability and navigability. The two-column layout was created using the float property, and the rest of the layout and design was defined in the CSS through element-, descendant-, class-, and id selectors.

When both the structural and presentational layers were finished, the web site went through a testing phase. Hyperlinks were tried out, and the web site was tested in a variety of browsers:

- Mozilla Firefox 1.5/2.0
- Internet Explorer 6.0/7.0
- Netscape Navigator 7.1/8.1
- Opera 7.5/9.0
- Safari 2.0
- Lynx 2.8.5

⁹A technique used to show users their location in web pages, i.e., Home> Documentation>FAQ

¹⁰This excludes the use of all deprecated elements and attributes that possible to use in the HTML 4.01 specification

¹¹World Wide Web Consortium

¹²“Lorem Ipsum” is a commonly used dummy-text

The web page was presented properly in all of the new browsers, as well as all of the older ones except Internet Explorer 6¹³. To see how a text based browser would present the content, it was tested in Lynx, and the result was very satisfactory¹⁴. In IE6, the layout was broken completely, with the content positioned below the rightmost column. Several approaches were used trying to fix this problem, and it was soon discovered that the problem could be fixed by setting the position of the columns as absolute, instead of using the float property. Then the web site was retested in the browser, and everything looked as it should. Since then, a bug tracker has been added to the menu, and some changes to the colors have been made. The final result can be seen in Figure D.2 in Appendix D¹⁵.

9.2.2 Search Engine Optimalization

Users usually examine the top 10 or 20 results when they perform a search [SMHM99, SWJS01], and the use of search engines was in one study found to be the most common method to locate OSS components [CLM⁺07]. Thus, it seems clear that the web site should be easily reachable by all major search engines when people search for related software, or as Stefan Landro put it: *“You have to be highly ranked on the search engines. To be on the top. It is very important that the web site is easily accessible through search”*. Vidar Langseid also emphasized the importance of ranking high at the search engines: *“We have always been aware of the importance of having a high rank at the search engines. eZ Publish is also a Content Management System, and thus searches for this must make us easy to find. We have tried to use good wordings in the URLs, the titles and so on”*. While he suggested that the use of good wordings can have an impact, Stefan Landro put it the following way: *“We don’t have any particular strategies on how to get a good ranking. A good reputation, many articles related to the project, that people blog about it, that posts in mailing lists link to it, and so on is important. Many links to the project will automatically give a high rank. At least on Google since they use the concept of hits and links in establishing the rank”*. In reality, the specific algorithm used by the major search engines are closely kept secrets, even though Google has revealed that their algorithm takes more than 100 factors into account [BI06]. What provides good ranking has however been analyzed, and some general guidelines can be followed to increase the chance of receiving a high rank, most of which were followed in relation with Keywatch:

- Submit the web site to major search engine databases.
- Put keywords in the title-text, headlines and visible body text [Thu03].
- Make sure title and alt tags contain descriptive and accurate information [Goo07].
- Avoid the use of special characters in the title [Thu03].
- Avoid the use of spam tactics [Thu03].
- Use metadata descriptions [ZD05].
- Write blogs, forums posts, and signature files referencing the web site [BI06].

¹³It was decided not to support IE5

¹⁴The web site presented in Lynx can be seen in Appendix D

¹⁵There are some inconsistencies due to scaling issues. The design can also be found at <http://cosiportal.idi.ntnu.no/portal>

- Create a site map, and submit it to relevant engines [Goo07].
- Submit the site to relevant directories such as the Open Directory Project and Yahoo!, as well as to other industry-specific expert sites [Goo07].
- Make a site with a clear hierarchy and text links [Goo07].
- Create a useful, information-rich site, and write pages that clearly and accurately describe your content [Goo07].

9.2.3 Web Site Content

The web site acts as a centralized platform for providing information about the project out to the public, as well as serving as an interface for the project tools. The information about the project must be presented orderly and the tools must be easily accessible. It should present a clear and welcoming overview of the project, and bind together the tools used in the project [Fog05]

General Content

The main page of the web site should give an introduction to the project, its features and the project license. It should be unambiguously clear that the project is open source and free. To not provide such clarifications might cause users to look elsewhere [Fog05].

It can also be a good idea to provide a mission statement at the main page to explain what the project does, and why it meets a need for prospect users and developers¹⁶. Provision of screen shots and demos (if applicable), can help visitors to understand what the project does, and act as a “proof” that the project can run.

Development Status

To attract users and developers, the current state of the project should be easily accessible, i.e., whether it is an unstable alpha, a beta, or a stable project. This includes providing a road map what will be included in future releases, or as suggested by Thomas Malt: *“Having a road map is very important”*. eZ Systems have also provided a road map what will be included in feature releases: *“Until recently, we have published plans for what will be included in the next release, within a six month perspective. We have changed it now, and we now use feature freeze instead. We release a beta, and then we spend a couple of months to testing and bug fixing before the final release”*. eZ Systems still provide a roadmap of the major features that will be included in the next release.¹⁷ Table 9.2 presents what the studied web sites provides related to development status. Most of the projects has an easily accessible road map, and three of them specifies how to get involved.

The web site should include a description of what contributions that would be especially welcome, i.e., feature requests. Interesting tasks that need further work should be suggested, and these should be simple enough to attract contributors, but challenging enough to be of interest [DM05].

¹⁶As seen in chapter 8, this might increase the chance of attracting people to the project

¹⁷<http://ez.no/ezpublish/roadmap>

Project	Development Status
netWiser	No information on development status available
Openbravo	Roadmap How to get involved
OpenEMM	Roadmap Known issues How to get involved
TYPOLight	Road Map Known issues How to get involved
Visual WebGui	No information on development status available
Webload	Road map

Table 9.2: Development status at studied web sites

Trolltech has a task tracker where the community can suggest new features and vote on which features and bugs they think should be prioritized¹⁸. Vidar Langseid explains what his thoughts on providing future requests and how they have been related to eZ Publish: *“We didn’t really list any feature requests on our web page. This is something that we plan to focus more on from now on, much due to not having quite as many outside contributors as we desire. The problem isn’t just to receive contributions, but to receive good contributions that are easy to implement and that don’t affect the other parts of the code”*. It seems clear that Vidar Langseid believes the provision of feature requests can be a valuable effort in attracting more contributions from the community.

Documentation

The importance of documentation have already been discussed in chapter 8. The software documentation should be available at the web site so that people can read it before downloading, as well as in the downloadable distribution [Fog05]. The documentation must be easily accessible from the menus and it should be made clear whether a certain document is intended for users or developers.

It was suggested in Chapter 8 that several variants of the documentation should be provided simultaneously, as each document has a special purpose. As seen in Table 9.3 all of the studied web pages provided a variety of documents that were available online. To ensure a more coherent development approach in the community, it can be a good idea to provide developer guidelines, as well as instructions on how to report bugs and submit contributions. The documentation should also point out important threads in the mailing list archives and forums.

Tools

The tools used in the project should be easily accessible. This includes information about the mailing lists and how to subscribe, accessible archives, and informing about IRC channels if such exist. Bug trackers, mailing lists, forums, and the revision control system should be integrated

¹⁸<http://trolltech.com/developer/task-tracker>

Project	Documentation Provided
netWiser	User Manual Installation manual Frequently Asked Questions API Documentation
Openbravo	Documentation Wiki providing: Technical Frequently Asked Questions Architecture Overview Installation Guide Developer Manual Localization - Frequently Asked Questions How to localize User Manual (English and Spanish)
OpenEMM	Frequently Asked Questions Documentation Wiki providing: Installation guides and General help
TYPOLight	Userguide Frequently Asked Questions Screencasts showing: How to install How to perform common tasks
Visual WebGui	Frequently Asked Questions Quick Start - How to perform common tasks Architectural Description Technical Manual
Webload	Frequently Asked Questions Quick Start - Requires registration, also available from SourceForge How to perform common tasks - Requires registration Users Guide - available from SourceForge Reference Guide - available from SourceForge Documentation Wiki providing: Architectural overview

Table 9.3: Documentation found at surveyed web sites

properly, so that information is posted to the mailing list and forum's when something happens in the other systems. It should also be clearly described who of the project developers are subscribing to the mailing lists [Fog05]. The source code repository should be browsable from the project web site, as this enables everyone to see the latest revisions of projects files, and to look at earlier revisions or to read log messages without using local clients. Generally, the hosting of these tools can be provided either by implementing everything yourself, or by using an external hosting solution.

9.3 Hosting the Web Site

Quite a large infrastructure is required for hosting an open source community. It is important to provide a reliable, scalable service for both those developing and those using the technology. When deciding upon how to host the project, there are two choices that can be considered; The use of an external “canned” hosting service or to implement your own solution. Implementing a well-functioning infrastructure yourself, demands much work and resources. This relates to ensure that the tools work together properly as their functions are interrelated and to maintenance.

Canned hosting offer prepackaged, templated web areas with accompanying tools providing most of the services required to run an open source project [Fog05]. The infrastructures provided by the foundations described in Chapter 3, provide canned hosting solutions, but are limited to the projects that meet specific criteria. There are also sites that provide such hosting services for everyone such as SourceForge¹⁹. All these sites serve as an external interface for multiple projects, providing several tools and services that are of importance when publishing a project. Due to their accessibility, such portals contain a large amount of projects, many of which are inactive. SourceForge is today the most widely used open source hosting solution, with a total of 150,109 registered projects as of June 4th 2007. It is owned by the Open Source Technology Group and provides a wide range of tools and services. This includes forums, mailing lists, revision control systems (CVS and SVN), and bugs- and feature requests tracker. SourceForge also provides the option to sell services.

There are some advantages associated with the use of a canned hosting site. These sites provide the required infrastructure for hosting an open source project, eliminating the need for implementing all the tools and maintenance of such infrastructures. It also provides server capacity and bandwidth for free. Bekk has used SourceForge for hosting their projects, due to reasons described by Stefan Landro: *“There is a lot of work in hosting a project, especially if the activity is high. You have to maintain the server and the communicational media. Too much work in relation with these issues when the activity became high, was essentially what made us transfer Middlegen to SourceForge”*.

Project	Use of SourceForge
netWiser	Does not use SourceForge
Openbravo	Bug Tracker Revision Control Forums
OpenEMM	Bug Tracker
TYPOLight	Bug Tracker
Visual WebGui	Registered, but does not use SourceForge
Webload	Revision Control

Table 9.4: Use of SourceForge by studied projects

However, the prepackaged configurations must be accepted as is, and might only be configured to a certain degree. Vidar Langseid explains their use of SourceForge. *“We used SourceForge in the beginning for its hosting services. We also had our own web page, but using SourceForge was a good*

¹⁹<http://www.sourceforge.net>

way to save bandwidth. And of course, we didn't have to worry about setting up and maintaining the infrastructure ourselves". Essentially, he summarizes the advantages of using such a site. However, eZ Publish changed their strategy shortly after: "We only used SourceForge for a short while before we created our own infrastructure with forum, bug tracker, and so on".

Table 9.4 presents how SourceForge is used by the studied commercial open source projects. It seems that using SourceForge as provider of a bug tracker is popular, and that some companies also use it to save bandwidth. All the projects except Openbravo had implemented their own solution for communication. This is probably because these are quite easy to implement, e.g., as opposed to bug trackers [ES06]. The prepackaged hosting solutions result in a lack of customizability and they might also cause reduced user experiences, i.e., related to the forums they provide. Knut Yrvin did not have much positive to say regarding the usability of SourceForge for non-developers: "For end-users that want to use the programs, SourceForge is useless. It is a web for developers. End-users just want to know where to find the program, point the mouse at a link for download, get some information about the project with a short description etc". While pointing out that it is a web for developers, he also essentially says that there must be another web page used for presenting the project to potential users. In the commercial environment, it might be better to use a "homemade" solution, as is also the strategy adopted by Linpro: "It is more natural for us as a company to create our own portal solution. SourceForge is just an infrastructural provider".

9.4 Promoting of the Project

The goal of open source projects might relate to have a project that is self-sustained, however to achieve this, the project must gain a "critical mass" of the market share [Kha00, BR03]. Achieving such a critical mass might however prove extremely difficult. The first step to attracting users and developers to an open source project, is to create an awareness of the project in the public. Stefan Landro provided some suggestions as to how marketing and promotion relates to open source: "Marketing of open source is generally like all other marketing. You can spend a lot of money for announcements in various media, you make sure to be interviewed about the product, you travel to conferences, you might sponsor conferences and have your own stand at the conference. You might also actively engage potential customers to present the project. But the cost of such marketing is high. It is mostly those that develop traditionally licensed software or those have open source as their main business area that do these things to a large extent".

Thomas Malt described the promotion of their open source projects, as well as what they would have done differently if they were to release a project today: "We have published quite a lot about Varnish in the Norwegian press. We have also been good at writing articles for international media, participated in international forums, etc. However, we have lacked a dedicated marketing strategy; haven't specifically decided upon performing marketing campaigns and so on. With most of our other projects, we have just created a web page, written an OK documentation, announced the project at Freshmeat and not really done much more than this. We have just made these tools available on the Internet. If we were to release a project today, we would probably spend more efforts toward marketing. We would probably have announced the project more heavily. We would have issued press releases to the media pertaining to the Norwegian IT community. We have recently taken more initiative in ICT Norway and their efforts in relation to free software. I am also interested in performing a more structured approach to the development of good web pages for our projects. Some

of the most important success criteria pertain to having a good web page with high usability, good documentation, an active wiki, and as complete API and user documentation as possible. Things such as having active mailing lists, providing credits to contributors, and releasing often at regular intervals are also very important”.

There exist little literature regarding specific promotional strategies for open source software. The adoption of open source software and its diffusion have rather been said to be influenced by its perceived intrinsic value and externality effects²⁰ [Kha00, BR03]. Some suggestions of promotional tactics were suggested in [ES06]:

- Present the project at conferences
- Create a proper web page.
- Post the project at major open source portals.
- Post the software for download at software portals.
- Issue a press-release.
- Use Wikipedia.
- Provide a news-letter.

It was decided to study the effect of these suggestions in relation with the release of Keywatch²¹, and based on the results and the interviews some changes and further options have been outlined in this section.

9.4.1 Developer Gatherings

To create awareness about the project before it is released, it is a good option to present the project, its rationale and why there is a demand for this project at gatherings of developers. This can relate to formal presentations at developer conferences as well as participation in gatherings where one can talk with other developers, and promote the project in such a way; Christian Schwartz: *“We participate in user groups such as Javabin and participate at Javazone. These are places where you get people to listen to what you have to say. We also have presented the project for another consultancy company, for students, and at conferences such as Software 2006”.* It is important not only to attract people to the project, but to attract the right people. Presenting the project at conferences can help with this, as stated by Stefan Landro: *“That’s what is difficult, to attract the right people. You need to have good contacts. If you haven’t presented the project at conferences, haven’t been able to create any publicity around it, then it will take more time to attract people. It is much more difficult then”.* All of the interviewees mentioned the importance of participation at developer gatherings in one way or another. Bekk has experienced an additional benefit to direct promotion from the participation of such gatherings: *“We had a presentation of CubicTest last year*

²⁰Negative externality effects include those from the dominant standards, while positive externality effects come from, e.g., the community of programmers

²¹Not all of these suggestions were possible to perform and measure

and there we met those hosting OpenQA. They were very interested in our project and wanted to host the project at their cite, and release it there. It is quite typical, conferences and workshops are very important to create interest and awareness about your projects". Media is sometimes present at such developer gatherings, which might lead the project to be presented in news articles if they manage to catch the media's interest. Also Linpro were present at developer conferences; Thomas Malt: *"Varnish hosts gatherings for the community. It is important to have developer conferences etc. occasionally. We present Varnish at a large amount of conferences, and we will also continue with such presentations"*.

While not actively participating much at developer gatherings initially, Vidar Langseid believes such participation is important. They have the last several years hosted their own gatherings as well as promoted eZ Systems and their projects through participation and presentation at external conferences: *"We didn't present our project much at workshops and conferences. Indeed, I think it would have been much harder to gain the same amount of attention today. Now there is so much open source software that it is difficult to get attention, to be noticed. When we released our project as open source, enthusiasts in a variety of companies "advertised" it in their companies, it spread from the bottom-up. Then, the managers contacted us. We now host gatherings every year. That is an important part of dealing with the community, and we have many that look forward to these gatherings. Not just from the community, but we also have large partner networks with many partners and some customers that show up"*. Trolltech also hosts their own arrangements: *"We have two arrangements each year, one in Europe and one in USA. We have also sponsored some arrangements, such as DebConf in Great Britain. Additionally, the community has their own gatherings in relation with Qt, where we provide sponsorship and are present"*. Trolltech are also active participants at several developer conferences. Obviously, companies such as eZ Systems and Trolltech already have succeeded in attracting a large community, and the hosting of such gatherings will not be feasible for small companies that have released an open source project. What it shows though, is that the participation at such gatherings are important for commercial open source adopters, independent of their size.

9.4.2 Announcing on related Web Sites

When the project has been released, it is vital that announcements are made at web sites where the project might be of interest. Keywatch has been developed using GWT Drag and Drop, and an email was sent to the author of <http://code.google.com/p/gwt-dnd/>, suggesting that the project should be added to the site. It became available from that site at 2007-04-18, two weeks before the official release announcement. Posting the project at this site has proven beneficial in the attraction of visitors to the Keywatch web site, as seen in Table 9.4.2. Thus, it is suggested that whenever a related technology site makes it possible to announce projects, this should be done.

Month	Keywatch Hits
April	77
May	202
June*	95

Table 9.5: Hits from posting Keywatch at GWT DnD web site

*Hits until 2007-06-16

It was suggested by [ES06] that posting the project at Freshmeat might prove beneficial. Freshmeat²² is a site where open source initiators can announce the release of their open source projects, write articles, as well as to announce when new versions have been released. Several announcements are made every day, and its visitors can then check out projects they find of interest. Freshmeat maintains ranking of the most popular projects, as well as of the most active ones, and might in this regard also act as an advertising channel to projects that achieve a high rank.

Linpro has seen a correlation between the posting at Freshmeat and the activity at their web site: “We have seen there is a link between the number of downloads and the activity on our mailing list, when we have posted the project at Freshmeat, and when people read about it at sites such as digg.com and del.icio.us. When Apache wrote about Varnish, then we could also see an increased activity, although not such a large increase in the amount of downloads”. The effect of posting the project at Freshmeat was tested out in relation with Keywatch:

Date	Freshmeat Hits	Keywatch Hits
2007-05-04	205	114
2007-05-05	129	88
2007-05-06	43	13
2007-05-07	51	26
2007-05-08	24	8
2007-05-09	14	1
2007-05-10	17	6

Table 9.6: Hits the first week from announcing Keywatch at Freshmeat



Figure 9.5: Hits from announcing Keywatch at Freshmeat

As can be seen from Table 9.4.2 and Figure 9.5, the incidence of hits directly after the announcement

²²<http://www.freshmeat.net>

peeked the first two days. Then it lowered considerably the first week and has remained stable at a few daily hits since then. Based on these results, it seems reasonable to suggest that announcing the project at Freshmeat is a good strategy to attract visitors to the web site.

It was suggested in [ES06], that posting the project at SourceForge could help in attracting users. It was already decided by Keymind that SourceForge should not be used to host the infrastructure. To verify whether having a project at SourceForge would have an impact on the attraction of users, the project was thus created there with some general information about Keywatch, and with a link to the web page. Derived from the server logs, there have been a total of 11 referrals from SourceForge. These results are considered insignificant, as most, if not all of these have come from either Keymind employees, or students actively involved with Keywatch. It should be noted that SourceForge provides rankings of the projects, and if SourceForge is used for hosting the technical infrastructure, this might provide an added promotional effect to already popular projects.

To provide some additional results, it was also tried to get the project posted at: <http://www.javalobby.org>, <http://java-source.net>, and <http://www.theserverside.com>. Out of these, only JavaLobby accepted the announcement, and the post at JavaLobby has caused 151 hits in the period from 2007-06-09 to 2007-06-16²³. This suggests that getting the project posted at sites where many people interested in software development and technology can help ensuring an awareness of the project in the community, and be directly related to the amount of hits the web site receives. eZ Systems has had similar experiences: *“We didn’t really do anything special to market eZ Publish. We used Freshmeat and SourceForge as external channels, hosting the project at SourceForge. We also posted the project at HotScripts²⁴, a page much like Freshmeat for announcing new releases and so on. I think it was Freshmeat that created the most interest”*. Taking into consideration that eZ Publish is a CMS, the posting at HotScripts can be related to the suggestion that technology-specific sites should be used as announcement channels.

9.4.3 Articles in the Media

A press release should be sent out to related media when the project is released. All of the studied projects except netWiser have issued press releases. All of the interviewed companies except Bekk have taken such an action as well, as a part of promoting their open source software. While the general media probably does not care much about the release of open source software, targeting the press releases toward related media that specialize in technology have been a successful strategy for Norwegian open source companies; Vidar Langseid: *“We did not spend any money on marketing campaigns in the beginning. We did however issue press releases, especially targeted toward the Norwegian computer-related web sites. They posted several articles”*. Also Linpro had much success in the press; Thomas Malt: *Things started happening very fast when the project was released. At least regarding attention in the professional press. We had some thousand downloads in the beginning, and then things stabilized after a while with a stable increase in the amount of downloads. It did, however, take some time before people began to actively participate in the mailing lists and so on”*. A press release draft was created in relation with Keywatch, but it has not been sent out yet, and thus the actual effect of such a strategy has not been measured. For

²³As the project did not get posted earlier, comparing of the results in relation with, e.g., Freshmeat is impossible

²⁴HotScripts.com is an Internet directory geared toward webmaster and programmers that compiles and disseminates Web programming-related resources

most companies, the most realistic initial option will be to get it published in local or national online news-sites. Here in Norway, sites such as <http://www.digi.no>, <http://www.idg.no>, and <http://www.itavisen.no> are good examples of such sites. However, if the company is very large and releasing formerly proprietary software as open source, major news sites, such as <http://www.slashdot.org> will probably also take interest in these news.

9.4.4 Wikipedia

In [ES06], it was stated that creating a Wikipedia article might help in attracting users to the project. To measure the effects of having such an article, a short objective one was created that explained the Keywatch software. This article was created at 2007-04-18, and contained a link to the Keywatch web site. The referral logs have later shown that this article has provided a total of 32 hits, although some of these are known not to be from external sources. Thus, based on these experiences such an article does not help much in attracting visitors to the web page, although it might provide some additional visitors. What might be an option in relation with Wikipedia, is to add the project to lists of tools that belong to a certain category, i.e., in the same way that SVN is listed as one of several Revision Control Systems.

9.4.5 Actively engaging potential users and customers

Active participation in online discussion forums and mailing lists can provide opportunities for creating awareness about the project as well as encouraging others to try it out. Bekk attributes much of the success of their Middlegen projects to such activities: *“I believe that most of the Middlegen adopters can be attributed to active participation at TheServerSite and through one of our developers actively participating in the XDoclet project”*. Christian Schwartz suggested that: *“Some people tip off the “right” people such as making deals that should recommend the project on their blog”*. Trolltech has pro-actively engaged users and developers taking several approaches: *“Participation in mailing list discussions, being interviewed by international online media. There were dialogs with developers”*. Knut Yrvin also related that actively engaging in sale has been more emphasized the recent years: *“In the beginning, Qt was published at the Internet, and that led to attention and customers. That was how the first sale was achieved; Trolltech has always been an Internet company. People contacted us and that went on for along time. However, after a while we realized that we have reached a saturation point, and now we actively seek out customers ourselves”*.

It was attempted to post about Keywatch in discussion forums related to OSGi and GWT, as well as those related to monitoring systems. However, it was difficult to keep objective and to find interesting sites of which the project could be subtly suggested. All the major OSGi sites were checked for options to such posts, but they mostly related to specific OSGi implementation, and posting at such communication forums were not performed because it could be considered as spam and thus create a bad reputation. There was more success related to the GWT, as it was found that several other projects using GWT had announced their projects. Information about Keywatch was posted in a one-time post at the GWT section of the Google User Groups at 2007-05-27, and 42 hits accumulated during the last days of May. An additional 13 referrals have come in the first two weeks of June. The rapid decline of hits is probably a result of the amount of threads accumulating at this site. Other options that can be tried out to create further attention to the project; such as

writing blogs about it, linking to the web site in email and forum signatures [Fog05].

While pro-actively spreading the word about your project in ways as described, Christian Schwartz emphasized that the effect can be greater if people in the community perform such actions: *“To be successful, someone has to spread the word of the project within the community. However, it is better if others praise your project than to do it yourself, as people have easier to believe outsiders than those that have created the project”*.

9.4.6 Partner Sites

Banner exchanges and web site exchange with partner sites might provide additional users to your web site. All of the interviewees except Bekk were found to have partner exchange programs at their web sites, although these linked to the partners’ web sites and not directly to the open source software. When asked specifically about the use of banner exchange, Thomas Malt answered as following: *“We haven’t had any banner exchange deals or similar, but it might be a good idea. We do have references and partners though, with links to the companies’ web pages. But we could have been much more focused on this in relation with the open source projects”*.

9.4.7 Newsletters

A newsletter is a subscription-based marketing tool for a business or association [Kin01] and is, e.g., provided by Openbravo, OpenEMM and Weblod. If your company already posts a newsletter about the company and its products, including information about the open source product will be a good idea. If not already having such a newsletter, it can be a good marketing strategy if it seems reasonable that people will take interest in such an option.

9.4.8 Paid Advertisements

Professional marketing related to paid announcements and advertisements in marquee space are costly [Kri05]. While companies selling proprietary software will often stick to direct advertising strategies, smaller open source initiators might not have the same opportunities. Thomas Malt explains why they have not concentrated much on paid advertisements: *“In the future, we will probably have a more specific strategy concerning marketing of our open source projects. I want to create open source projects out of more of our internal systems. But it can be risky, this isn’t something we live of directly, and a software company needs to create revenues. We cannot spend millions on something of which we don’t get anything in return”*. Vidar Langseid also described the problem with paying for advertisements: *“The sale aspect is much more important today than it was when we went open source. We probably would have paid for marketing if we were to release eZ Publish today. But of course, that is also a problem. Small companies cannot afford large marketing campaigns. We would probably only have targeted the nearby business”*. Trolltech pays for advertisements, but they have focused their advertisement toward “niche” media, as Knut Yrvin put it: *“Yes, we buy advertisements in certain media - fitting media as I like to call it. We have among other things recently bought an advertisement in “Friprog-magasinet”²⁵. Thus we target our*

²⁵Norwegian magazine about free/open source software

marketing toward media of interest to developers, niche media”.

9.4.9 External Effects

A major source related to the promotion of open source projects can be attributed to what has been described as positive network externality effects in the literature [BR03]. Mozilla is a great example of community promotion, and is a counterexample to the long assumed asymmetric relationship between the corporation and the consumer, where the consumer is viewed as a passive recipient of marketing messages from the company [Kri05]. The Mozilla community organized several web sites, including a site of which its purpose was specifically directed at promoting the project [Kri05]. Positive reviews were written in blogs and at third party web sites, email signature files were used to provide a link to the download site, and banners were posted on individual web sites. These promotional activities focused on the benefits of Firefox over Internet Explorer, and thus the concept of meeting a demand in the community was met. Trolltech also described the community as being responsible for most of the advertising effect when asked about what they had done in such respects: *“The answer is very simple, dual licensing was the only thing we did. We posted the project at the news-system that was widely used for open source at that time. We wrote that Qt was available for free, and people tested it out. The community found out how good the system was and rumors spread within the community, attracting more and more people”*. The collaborative effort of a large number of bloggers and forum participants can also influence the ranking at search engines such as Google [BI06], and the project can be promoted through personal contact.

Although the amount of community promotional activities has been minimal in relation with Keywatch, some examples of such activity has been performed²⁶:

- The project has been added to the database of <http://www.linux.softpedia.com>, resulting in 25 direct downloads, and 30 hits at the web site
- The project has been recommended in the forums at <http://www.macuser.de>, resulting in 31 web site hits
- The project has been added at <http://del.icio.us>, resulting in 4 web site hits

While this activity has not yet proven highly significant, it provides some evidence that even small communities can help with the promotion of an open source project.

9.5 Summary

This chapter has presented several issues related to the release of an open source project. It has described the tools commonly used in open source projects, and why providing such tools are of importance. It has also discussed whether to use a canned hosting solution or to implement the infrastructure yourself. Additionally, the chapter has gone through the creation process of

²⁶Student participants and Keymind employees have provided assurances that they have not been behind these posts

the Keywatch web site, and discussed certain measures that can be of importance when creating such a site. It has described what the web site should present together with a rationale for these suggestions. Finally, the chapter has described promotional activities that can help to create awareness and interest of open source projects. The effect of several such activities have been measured in relation with the Keywatch project, providing some evidence of which activities are most successful.

Chapter 10

Community Management

The last several chapters have described the process of commercial open source initiatives. Starting with the identification of the rationale and going through the process of releasing and announcing the project. This chapter addresses research question 3, and goes through the final aspect of such a process; management of the community after the project has been released. It describes the importance of ensuring continual activity and good communication with the community. It also covers leadership and organizational issues, what contributions “expected” to receive, and the importance of providing credits to contributors.

“The community and the interaction between volunteers and professionals create innovation. It creates sparks, issues, solutions. This is innovation” - Knut Yrvin, Trolltech

10.1 Communication- and Development Activity

The activity in the discussion forums, the number of changes to the source code, and the amount of bugs reported can signal an open source project’s health [SFT07]. It seems reasonable then, that these factors can help to attract a community to the project. The community may consider projects with high activity to provide a better chance to have potential for success.

10.1.1 Communication

Communicational platforms with high activity and quick answers to community questions and suggestions can help with the attraction of a community [Fog05], and can be an important factor in sustaining the community. The interviewees were asked how the interaction process with their communities were handled, and Thomas Malt described their process as following: *“We have internal meetings, and work together at the office. But all solutions and similar issues are discussed in mailing lists, providing a written documentation of everything we do. I am certain that having open communication, to show what is discussed internally, and active participation on the mailing lists can help attract contributors”*. He emphasized the importance of opening up the communication with the community, not only to answer the community’s requests and inquiries, but also

to publish what the company has discussed internally. That this might prove beneficial has also been suggested in the research literature [Stü06] [Fog05]. Stefan Landro also suggested that to have active discussion forums and mailing list can help attracting people to the project: *“Having an active forum and mailing list is also important, as it shows that there is activity in the project, that people take an interest in it”*.

eZ Systems has decided upon another approach; Vidar Langseid: *“We didn’t post our internal discussions on the forum, and we still have discussions about eZ Systems that we don’t post publicly. We have discussed this internally, and there are of course both advantages and disadvantages to such an approach”*. Vidar then went on describing the importance of answering questions: *“We have participated actively in the discussions on the forums, and this is something we still do. It is important, and we should probably have spent even more time on this than what we do today. Answering questions is especially important in the beginning, because then you are the only one that knows how things work”*. Although answering questions early is important, eZ Systems has experienced such a large amount of communication that this is no longer feasible: *“We were always quick to answer questions, and were very focused on this in the beginning. It is different today, much because there is too much going on to answer everything. It is mostly the community that answer questions these days”*.

Christian Schwartz seems to agree with Vidar Langseid, and suggests that in larger projects, questions and suggestions have to be prioritized: *“It is important that people with high knowledge of the project answer questions, however, they don’t have to answer all the stupid questions that are posted. They should answer the difficult technical questions, and let those less knowledgeable answer the others”*. What questions that are to be answered, can also relate to the business model adopted by a company. If the company sells support, they might be more unwilling to provide in-depth answers to questions from the community. Vidar Langseid explained the dilemma of what to answer in relation to eZ Publish: *“There is also another problem related to what one should answer and what should be transferred to the support section. In the beginning, we answered all questions, and if we get a concrete question by email, we usually answer quickly. However, only if the answer is easy to provide. We don’t go in depth to provide answers to time-demanding questions, but rather provide pointers or refer to some documents. We typically say: We have a community, and you can ask questions on the forum, maybe someone there can help. We also explain that we provide a professional support services, where they can get consultancy directly related to their specific problem. Of course, that too can cause problems. Some people think that we don’t answer them just because we sell support”*.

As the activity on the mailing lists and forums can help to attract users and developers, it is a good idea to post all company-internal discussion that relates to the development of the open source projects in these media. Indeed, using, e.g., the mailing list as the natural medium for such communication, even when the company developers are located in the same room might prove beneficial. However, what is most importance to achieve activity is to get people to download the software, as Stefan Landro described: *“The first step is to get people to download it. In my experience, questions will come as long as the project get downloads. You have to be prepared to spend time helping and answering questions, especially early on. This is also typical, as new bugs will often be discovered when the project is released to the public”*. When these questions come, they should be answer as quickly; Thomas Malt: *“To maintain a good interaction with the community is vital for success. It might be possible to get downloads, and if the software is good enough, then people will work on it in any case. However, answering in a friendly tone with little delay is very*

important". It is important that the core developers are active in the discussions, and that they are highly available. While Stefan Landro suggested that questions will come automatically when the project is downloaded, there has only been one external subscriber and poster to the Keywatch mailing list as of 2007-06-16. This can suggest that the amount of downloads and the size of the community are too low to attract more posters, as well as that people find the software to work without problems, and thus do not have any incentive to ask questions.

10.1.2 Web Site

Just as showing high activity in the discussion forums can help showing the health of the project, regular updates to the web site will help visitors to understand that things are happening. Thus, such should be done regularly, unrelated to the activity in the discussion forums. eZ Systems ensures regular updates to their web site: *"The web site is regularly updated. The documentation part is difficult though, you can never have too much documentation. There are still some problems to have all the documentation ready at the same time as the software is ready. We want to be better at this in the future"*.

10.1.3 Development Activities

The activities related specifically to the development; how often new versions are released, and how often code changes are made to the repository, can be seen as a measure of the project's health. Frequent development releases which include code that have recently been submitted, can provide a significant motivation for the developers [Fit06]. All of the interviewees agreed that such activity was an important factor.

Regular development activity should occur, thus, developers can see the daily changes to the source code. This process can differ from project to project, but the principle of daily builds that was described in Chapter 5 is commonly used. Knut Yrvin described the benefits of this approach: *We have daily builds. The frequent updates make it possible for people to see if their contributions have been added to the code base, and to provide feedback and be aware of what is going on*". Also Stefan Landro emphasized the importance of having an active development process: *"To have an active project is very important. Frequent releases show that there are regular commits to the code base"*. To ensure that the frequent releases and daily builds have been tested for the most obvious bugs, the use of unit tests is a great way to help ensuring this; Thomas Malt: *"Unit tests are great to assure the quality of distributed development efforts. There should be tests alongside all accepted patches, and all old tests should pass. It is important to have an automatic build script to be able to have frequent releases. A project might easily end up with several weeks spent on bug fixing and testing if such measures aren't taken"*.

Providing regular builds and activity related to the source code, the open source provider should also publish road maps specifying what will be included in feature releases. Bekk provides road maps for their open source projects; Christian Schwartz: *"We plan what to include in the project regularly. Probably once a month we create a list of what should be finished the next month. This TODO list is available for everyone"*. These releases should come at regular intervals, as emphasized by Thomas Malt: *"It should never take more than six months between each release, i.e., KDE has probably lost many users to Gnome because of more activity there. Smaller projects should probably*

release new versions once a month, perhaps every second month at most. Providing road maps is also very important. We have began using Scrum, and focused on an iterative development process internally. This incorporates the maintenance of a road-map, a TODO-list, known as a backlog in Scrum. This contains a list of features that are to be implemented and is a great tool for the developers". eZ Systems has generally worked with a six month perspective: "We have generally worked with six months to a yearly perspective pertaining to the community. We have of course also had plans extending these time frames, but not in the public. It has been like that for a long time. In the beginning we didn't really plan much, we just went on day by day. If we received a cool request, then we implemented it. It was very ad hoc. "

10.2 Leadership and Organization

The inter-relationship between open source companies and communities seem to comprise a set of tensions and inconsistencies that can lead to certain managerial issues [DM05]. These tensions can relate to questions of ownership, decision rights and control. The company's desire to guide the project to meet its own needs must be weighed against providing incentives for the external community to join and contribute to the project [WM05]. The Trolltech licensing model was the source of much conflicts with the community, before they adopted the General Public License: *"Our dual Licensing model developed over time. The source code was always available, that wasn't the problem. However, we did not use GPL. Most didn't care about this, but the choice of licenses are important for some. There were much discussions when Qt wasn't truly free software, and there were always questions like "When will Qt be free?". Thomas Malt puts some perspective into such issues: "There are both conflicts of interest and political conflicts. Some people in the open source world are very idealistic, being almost dogmatically concerned with keeping everything free and to avoid everything commercial. In the real world, everything cannot be completely free or gratis, at least not for us".*

The choice of license in certain cases might be the source of tensions within the community. Tensions can also be related to the leadership by a commercial entity not internalizing enough of the same objectives as the open source community [LT05a]. If the company maintains too much control, the existing community might choose to abandon the project, to create a competing product, or to create a competing fork. This was, e.g., seen in the KDE project, where a competing project was created with the intention to replace Qt as the GUI toolkit¹. When Trolltech changed their licensing, the competing project was abandoned. However, it is important to balance the amount of control, as if the control is too little, the effect for the company might be small or even counterproductive [DM05].

It is important that the organization provides high quality leadership, and character traits of the initiators such as commitment, experience, assertiveness, patience, and good communication skills can all help to ensure that this position is retained [Stü06]. Stefan Landro: *"It usually works in the way that the initiator of the project maintains control".* This has also been shown in the research literature related to meritocratic structuring [BR03]. All the interviewees claimed that they were indeed the final decision makers. Knut Yrvin described the decision-making process in the following way: *"We own and decide what happens with Qt. We have an internal process*

¹The story behind these conflicts can be found at: <http://www.kde.org/whatiskde/qt.php>

and go through suggestions, coming either from the community, customers, or ourselves. We have dialogs with the community regarding new functions, what they would like to have available in Qt. These suggestions are sometimes added to the project plan. We then have to go through these to decide where to put the development resources". Also eZ Systems make the final decisions related to their projects "We make the decisions, to put it that way. But of course, we have to take both the community and the customers' requests into consideration. It is always a dilemma what to implement. We usually implement some community requests, and a of course a larger amount of customer requests, as this is where revenue is created. These requests will often be the same, and then it is an easy decision to make". While both these companies stand for the final decisions, they make sure that the community's wishes are followed as much as possible.

10.2.1 Commit Access

Companies will retain a leading position in their projects, and they should not allow direct contributions to the code-repository without reviewing the code or making sure that trusted volunteers perform this work. Trolltech is very restrictive in this regard, making sure that all the source code are added by their employees, to ensure that they maintain ownership of the source code. This is a prerequisite for adopting the dual licensing model with GPL and selling of proprietary licenses: "Only those employed by Trolltech have commit access to CVS. People send patches to the mailing list, and we have to control the quality of these. We have to rewrite to make sure that the fixes work with the entire system, and not only within the context where the submitter uses it".

In other commercial initiatives, contributors from the community will often gain access if they have proven their skills, pertaining to the meritocratic structuring often found in open source projects. Stefan Landro said it the following way: "One usually doesn't become a committer for no reasons". He then went on showing an example where a person has been proposed as commiter. The proposal described the contributions the proposed commiter had provided, referring to the patches and features that he had created. eZ Systems has some external contributors: "There are external contributors to the projects, but not as many as we would have liked. They mostly achieve this status through being active participants in the community and providing good contributions. They must provide several patches to achieve this status. Then we know that they provide useful contributions to the project".

10.2.2 Symbiotic Relationship and Avoiding Conflicts

It can be a good idea for commercial actors to seek external participation in the creation of governance mechanisms [WM05]. When doing so, they must walk a fine line between asserting preferential decision rights and losing all influence over the project. A well functioning governance model should enable the sponsor to influence the project's evolution through pluralistic support [WM05]. Company involvement might to a certain extent obstruct the possibility for a community to have the desired ownership [DM05]. The main target then, should be to achieve what has been referred to as a "symbiotic relationship" with the community [DM05] [LT05a], i.e., that both the company and the community derives benefits from their interactions. Legitimacy to influence the development cannot be obtained only by having a formal role in the company, status must be gained by providing good contributions [ES06].

However, from the interviews it seems that most conflicts with the community are related to what will be included, and indeed some users of the software might try to demand functionality without contributing themselves. Thomas Malt described this in the following way: *“When you create an open source project, you do the users a favor. People cannot demand functionality if they don’t contribute themselves. A good solution might in some cases be to promise a certain amount of monetary reward for the implementation of features. For a commercial actor, it is important to find a proper balance between open source and commercial interests. Obviously, open source developers have to earn money just like everyone else”*. He went on explaining the importance of the company actively participating in the development: *“If a company wants to control what is to be included in an open source project, they have to participate actively. This might be done either by using your own resources, i.e., develop the software full time and implement what is requested, or such services may be bought from someone else. There are no guarantees, unless one is willing to spend resources on development, either internally or to pay someone”*.

There will always be the possibility for tensions between the commercial actor and the open source community. It might be especially difficult for a company to gain acceptance for using community-developed functionality in their commercial applications, such as, e.g., in the proprietary licensed code of companies adopting a dual licensing approach. This can be negated by maintaining a good relationship with the community. Proper leadership and the organizing of social events can ensure a good relationship. Face-to-face meetings can improve the social relations and make it easier to gain acceptance for commercial use of knowledge created in the community [DM05]. Some companies might try to leverage the community while keeping direct involvement in the development of the communal resources to a minimum, described as a “commensalistic” approach [DM05]. Such an approach is not recommended, and the importance of spending resources on development and on maintenance of the community to sustain it, seems clear. Certainly, no company should focus only on their own benefits while not adhering to the community’s wishes to any extent. This might cause the company being perceived as a “free-rider”, causing a negative advertising effect, and to the creation of competing forks. Being perceived as a “free rider” can also act as an additional motivational factor to make the forking project out-compete the original one.

10.3 Providing Credits

Providing credits to the contributors from the community, whether these relate to fixing of bugs, implementation of new functionality, or the creation of documentation, are important aspects of leading an open source projects. Christian Schwartz emphasized the importance providing credit: *“We give credit to everyone that contribute in our projects. That is normal and a proper custom. The difficulty might be in deciding upon which granularity to use, how to provide the credit. I.e., whether one should provide credit according to features added and specific contributions, or just list everyone that have contributed”*. There are several approaches to how the credits are presented. It has been suggested that what is most viable is to list all that have contributed to the project at the same level, as potential contributors might be discouraged if they find that, i.e., a core developer has implemented almost everything [Fog05]. Thomas Malt agrees: *“We don’t divide credits in any way. Everyone that have contributed are listed alphabetically, no matter how much or little they have contributed. Some projects divide into core developers, external developers and so on, but I think this is a bad idea”*. eZ Systems has also seen the importance of providing credits:

“We haven’t been very good at providing credits up until now. Last year, we implemented a new system that automatically gathered information about the contributors from the code, as this is also documented in the source code. The contributors are listed at the web site, showing which part of the project to which people have contributed”. Generally, one should always provide credits as this is an important factor related to individual motivations. Examples of motivations, can be related to future monetary rewards and the belonging a community as were described in Chapter 5.

10.4 Contributions from community

There are several benefits that can result from a good relationship with a well-functioning community. The company might receive suggestions for improvements, bug reports, and direct contributions such as development of new functionality, help with the documentation, and fixing of bugs. Knut Yrvin described how the community has helped with the development of Qt: *“The community provide much information in their bug reports. Often this relates to the code quality, e.g., a function tat does something other than what was expected. Also, the documentation might be unclear in certain cases. Of course, there are also true errors, i.e., that the software doesn’t work properly. This is the job of the support team; to separate what are true bugs and functions that should be improved. Our general experience is that since very many people use Qt voluntarily, we receive a lot of feedback regarding the quality of the product. About half of the feedback we receive is indeed from the community, perhaps numbering around 2500 between each release. All this feedback mean that we can deliver an industrial product significantly faster than our competitors. In fact, we get help from others to make our product better. This is the main benefit we receive. The second benefit is that many people that choose to use the project voluntarily “fall in love with it” and bring it to their workplace. Very many of those working with free software also do so motivated by what they learn and the knowledge they gain to be used professionally. It isn’t true what people say about free software not being commercial, just because it isn’t as easy to sell it. That’s why we have dual-licensing. We earn money by using a dual license so that we can also sell the product. We implement everything we get from the community ourselves, and this is because this is a program library . If a developer corrects an error, or fixes a bug, they might solve a problem locally in the library. However, this can often cause side effects that can cause new problems, leading to the introduction of new errors. To avoid this, and not lose to lose copyright of what is ours, we fix all faults”.*

eZ Systems takes advantage of value added contributions from the community, related to the development of additional modules and plug-ins: *“Small bug fixes and similar can be posted in the mailing list, but we prefer that it goes through the bug tracker. We have also created a site on our web page for uploading of plug-ins and modules. People can create a plug-in for their own need and upload it there. When this plug-in matures, we will take a look at it, and consider whether to include it in the main product”.*

External promotion is as we saw in Chapter 10 an important activity as communities will eventually die out if no new members replace those who leave [BSKK02]. The existing communities play an important roles in the promotion of an open source project. Promotion through word-of-mouth, by posting references to the project web site in other groups, through project reviews, referring to the project on personal web pages and email signatures, and voting on a project are all examples of such activities. In addition, activity itself has importance to building a community, as has been

described earlier in this chapter. The creation and consumption of content, the asking and answering of questions in discussion forums, and provision of content to the source code repository, will in successful projects to a large extent be provided by the community itself. Without participation, few of the characteristics that are beneficial to belonging to a community will come about [BSKK02]. Thus, all of the issues related earlier will help to attract an initial community. As long as the relationship with the community remains “symbiotic”, this community will in turn create positive external effects. Indeed, promotion through the community were by all of the interviewees attributed to being the most important reason as to why they had succeeded in attracting a large amount of users and external contributors.

10.5 Summary

This chapter has discussed management of an open source community. It has provided suggestions for how the community should be handled, and focused on the importance of ensuring an active project. It has described traits related to the leadership and organization of open source companies, focusing on how commercial actors must ensure a good relationship with the community. The importance of motivating the community and to provide credit has been discussed, and it has provided a description of how communities can contribute to an open source project.

Part IV

Discussion

Chapter 11

Evaluation of the Project

This chapter contains an evaluation of the project. It contains an evaluation of the research questions and the answers to these, as well an evaluation of the limitations of the study and the validity of the findings.

11.1 Evaluation of Research Questions

The following problem description was provided for this project:

The student should study processes related to commercial release of open source software and identify criteria that are important for achieving a successful project.

Three research questions, together with elaborating subquestions were created, focusing on preparations, the release process itself, and the period after the initial release. To provide an answer to the problem, it was decided to adopt a case study approach. This allowed for a qualitative study of the release processes found in commercial open source projects, and enabled an in-depth description of these processes based on first-hand experiences.

11.1.1 Research Question 1

What considerations and preparations are important to make in advance when releasing an open source project?

The first research question relates to measures that should be taken in advance of releasing an open source project. Answers to this research question were derived from the literature, and corroborated with what the interviewees described related to what they have done, and what they think of most important.

RQ1.1

What is the rationale behind commercial release of open source software?

Chapter 7 provides an answer to this question, describing the rationale behind commercial release of open source software.

Achieving a better product

This motivation is a commonality for all open source actors, but companies that do not derive their revenues directly from open source software might find this as the initial rationale behind the release of software as open source. Examples of such companies are Bekk, Linpro, and Keymind as they have developed software for internal use, and hope to receive contributions from an open source community to make these products better.

Creating Revenues

There are several business models, which companies can derive revenues directly from releasing their software as open source. Companies such as Trolltech, Linpro, and eZ Systems have seen the potential to earn money from the sale of support, services, and through the adoption of a dual licensing strategy. Indeed, achieving an increased revenue stream is also an underlying motivational factor for companies such as Bekk and Keymind, although this effect comes indirectly from effects such as increased productivity by achieving a better product.

RQ1.2

What measures are important to consider before a project is released?

Chapter 8 describes several measures that are important to consider before the release of open source software, and how companies relate to these, answering this research question. **Project**

Properties

Several project properties that should be present are described below:

- The project name should not infringe any trademarks. A name that is similar to other software or products might in general have a negative impact.
- The code must work properly. The project should be easy to both install and run, and some basic functionality must be present.
- The project should meet a demand in the community. Companies should make sure that the project provides something new, or something that might provide more value than competing products.
- The architecture must be suitable, including a clean and structured code base. A modular architecture can be viewed as a prerequisite to attract external contributors. At the same time, it is important to adhere to open standards and protocols. The software components must be legally combined to ensure that no licenses are violated.
- Providing good documentation is important. While the provision of complete user manuals might not be necessary upon an initial release, information on how to install and perform common tasks should be present. There must also be some documentation available for potential developers, and an API is especially important.

- The choice of license might have an impact on the project's success. Target users, and competing and complementary products should be considered in the choice of a license. The choice of license might impact which business models that are viable.

11.1.2 Research Question 2

What impact will the processes applied when releasing software as open source have on the success of the project?

The second research question relates to the release of open source software, and how the processes can impact the success of the project. An answer to the research question is provided in Chapter 9.5, and is based on findings from the literature, interviews, and surveyed web sites, as well as through participation in the release process of an open source project. 9.5.

RQ2.1

What comprises a good technical infrastructure, and how important is the provision of a good infrastructure to attract a community?

There are several tools that can help enhance distributed collaboration, and the purpose of the following tools have been described in Table 11.1.

The provision of an infrastructure enabling collaboration is vital in any open source project. What is of most importance is to provide either a mailing list or a forum for communication. A revision control system, and a bug tracker can also greatly enhance the collaborative environment, although it is possible to receive contributions and bug reports through mailing list or forums as well. It seems that mailing lists are more effective for coordination of development efforts, while forums are more user friendly for the non-developers. What tools to use depends on the project itself. A combination of both forums and mailing lists can be viable if the communicational activity is high enough. Wikis can also be a good tool to contain the documentation, and to make it easier to receive documentation contributions from the community.

RQ2.2

What is the importance of providing a good website for an open source project?

The provision of a good web site is important as this site acts as the project's external interface to the community. The web site should provide good usability with a well-functioning layout and design. The web site should be created in layers, keeping design, structure, and dynamic presentation separated. It is also of high importance that the web site is tested properly, to ensure that all hyperlinks are functional, and that the web site is presented as intended commonly used web browsers. When creating the web site, optimization for search engines should be considered and general guidelines for achieving higher rank in the search engines should be followed.

Tool	Purpose
Mailing Lists	<ul style="list-style-type: none"> -Communication -Coordinating the development effort -Creating awareness of the development process -Asking questions, getting help -Reporting bugs, providing suggestions, providing code contributions
Forums	<ul style="list-style-type: none"> -Communication -Coordinating the development effort -Creating awareness of the development process -Asking questions, getting help -Reporting bugs, providing suggestions, providing code contributions
IRC	<ul style="list-style-type: none"> -Communication -Creating a greater sense of belonging to a community -Lightweight discussions, getting help
Instant Messengers	<ul style="list-style-type: none"> -Communication -Most viable for one-to-one communication
Revision Control Systems	<ul style="list-style-type: none"> -Synchronizing development efforts -Enables distributed development efforts -Provision of source code
Bug Tracking Systems	<ul style="list-style-type: none"> -Reporting bugs, suggestions, issues -Keeping track of development efforts, state of bugs, and issues
Wikis	<ul style="list-style-type: none"> -Presentation of documentation -Collaborative creation of documentation

Table 11.1: The purpose of tools used in open source development

As the web site acts as the external interface to the community, the most important information must be easily accessible. The front page should include a short description of the project and its features and benefits, as well as making clear that the project is open source and what license that is used. To increase the chance of receiving contributions from the community, the web site should contain a road map showing the feature plans, and suggest improvements to the project of what kind of help is wanted from the community. The web site must also link to the tools that are used and to the documentation.

The infrastructure can either be implemented manually, or in a canned hosting solutions such as SourceForge. It is suggested that the web site itself is registered on a specific domain, and that forums, if used, are implemented manually. Relating to mailing lists, bug tracker, and revision control, the choice is more open. Manual implementation requires quite a lot of work and maintenance, but provides the benefit of more control and customization.

RQ2.3

What specific promotional activities are performed by commercial open source actors, and what is the effect of such activities?

Announcing and promoting the open source project is of high importance to create an awareness of the project's existence and to attract potential users and developers. The promotional activities and their purpose have been outlined in Table 11.2:

Promotional Activity	Effect
Developer Gatherings	-Create awareness and interest -Can be a good place to establish a larger network of contacts
Related Sites	-Probably the most important promotional activity -Create awareness and interest -Enhance visibility
Articles in Media	-Several companies have succeeded in getting articles published -Create awareness and interest -Especially useful to attract users that otherwise would not have actively searched for open source software
Wikipedia Article	-Might help to attract people looking for specific tools -Does not seem to be very effective as a promotional activity
Actively Engaging	-Creates awareness and interest -If one is already active participant in discussion forum, then it might be even more effective due to increased trust
Newsletters	-Provide information to those who are already aware of your company and/or project -A good strategy to remind people of a project's existence and as an information channel
Paid Advertisements	-Create awareness and interest -Can specifically target people that are more likely to be interested than the general population -Advertising in marquee space is expensive
External Effects	-Probably the most effective means of promotion -Relies on having already created an interest and awareness of the project -Relies on the provision of a good product

Table 11.2: Promotional activities for open source projects

11.1.3 Research Question 3

How do commercial actors manage and interact with the open source community?

The third research question relates to how commercial actors manage and interact with the open source community, and what are of importance in this regard. Answers to these questions are presented in Chapter 10.

RQ3.1

How important is the interaction and project activity to attract and sustain a community?

Interaction is vital to attract and sustain a community. Questions should be answered in a friendly tone and with little delay. If no response is received, then people might give up on the project. Additionally, it is important to show that there is activity in the project. The communication activity, providing regular web site updates, and the provision of source code at regular basis provide “proof” that the project is healthy as well as tentative evidence that such activities will continue. Frequent releases can also act as a motivating factor to contribute to the project, because people can see their contributions being adopted to the project.

RQ3.2

How do companies accommodate and adapt to the community in order to attract and sustain it?

How commercial open source actors accommodate and adapt to the community vary to a large extent from company to company, the commonality being that most companies consider the community when decisions are made. Some aspects found in the study has been listed in Table 11.3:

11.2 Limitations and Validity Threats

11.2.1 Limitations

There has only been one investigator gathering data for use in this project. Multiple investigators enhance the creative potential of the study and team members will often have complementary insights which add to the richness of the data. The convergence of observations from multiple investigators also enhance the confidence in the findings [Eis89]. This limitation has been reduced through reviews of the project, its design, and the interview questions by the supervisors, however it may still be present.

There have also been some challenges that have had impact on the final forming of the research. Indeed, it is recognized that case study research will often lead to a change in the research approach during its course [Eis89]. Part of my study was intended to validate the results from the depth study by taking an active part in the Keywatch release process. Due to external factors, it was not possible to validate these in their entirety from this process, and it has only been possible to assemble knowledge from the creation of a web site and the promotional process in relation with Keywatch. The web sites that were selected for study was not approached as in-depth as originally

Community Management	Effect
Leadership	<ul style="list-style-type: none"> -All the companies acted as the final authority -Human traits such as patience, commitment, experience, assertiveness, and good communication skills are important. -The meritocratic structure can still be seen in commercial open source initiatives. -To sustain the community, the company must still adhere to common requests. -Lack of adherence to the community can lead to creation of competing forks.
Implementation	<ul style="list-style-type: none"> -Whether the external community participants get commit access to the code repository varies. -Several community requests are usually implemented to the project. -Companies favor paying customers when deciding on what to implement. -Individuals in the community cannot successfully demand functionality without providing contributions.
Conflicts	<ul style="list-style-type: none"> -Conflicts will often occur. -Can relate to choice of license. -Can relate to what is implemented. -Can relate to individuals perceiving the company more as a sales organization than an open source initiator. -Do not seem to be much different from projects initiated by the open source community.
Credits	<ul style="list-style-type: none"> Important motivational factor. Whether it is provided varies. How it is provided varies. It is recommended that no distinctions are made between company-internal and external contributions.

Table 11.3: Commercial accommodation and adaption to the open source community

intended, and they were rather used as a supplementary source of data to describe the presentation of commercial open source projects.

11.2.2 Threats to Conclusion Validity

Conclusion validity relates to the validity of the relationship between treatment and outcome [WRH⁺00], i.e., whether the correct conclusion can be drawn based on the the research process.

Low Statistical Power

Only four interviews were conducted, and the statistical power is thus very low. More thorough research, i.e., the assembly of quantitative data will be needed to validate the guidelines presented in this thesis.

Reliability of Measures

Due to inexperience in performing interviews, factors such as poor question design and poor question wording might have led to misunderstanding, and a certain lack in the reliability. However, this validity threat has probably been negated to a large extent due to the general interview guide approach, and through probing and further specification of what was intended to be answered in each question where the interviewees did not understand the intent of the interview question.

Fishing and Researcher Bias

Searching for a specific result is a threat to the conclusion validity [WRH⁺00]. While there have been a high focus on not to let researcher bias influence the findings, there might have been, e.g., some leading questions during the interviews that can act as a threat to the conclusion validity. In addition, bias in the reviewed literature, and bias of the interviewees might have an impact on the result. Through studying external project web sites, and through reviewing literature that oppose the views found in the literature, this threat has hopefully been negated as much as possible.

Improper Combination of Findings

The literature has been reviewed for objectivity, and most of the literature used have been taken from well-recognized publications and peer-reviewed material. Several sources have been studied to identify commonalities, including the study of a large amount of literature, literature from a variety of research fields, interviews, and surveyed projects. The threat of improper combination of findings in the pre-study have been negated to a large extent due to meta-analysis, making sure that several publications have stated similar findings. Lack of validation of the research design in these publications, and whether combination of the results thus were valid, might have been lacking to a certain extent. Relating to the general guidelines, the finding in the literature literature were further corroborated with data from the interviews and findings from the surveyed project web sites. It is believed that this threat is negated properly.

Citation Bias

The literature selection process has to a large extent been based on heavily cited sources. This might be a threat to the validity of the project, i.e., due to the large amount of the research literature being based on articles such as “The Cathedral and the Bazaar” [Ray02].

11.2.3 Threats to Internal Validity

Internal validity relates to finding of a causal relationship when there is none [WRH⁺00]. Obviously, as this research is qualitative there have not been any statistical analysis implying a causal relationship. However, internal validity is still important to consider in qualitative research [Mal01]. Findings and interpretations must be questioned, and issues such as the reliability of the interviews might impact the internal validity implying a causal relationship when there is none.

Selection of Cases

Depending on how the subjects are selected from a larger group, the selection effects can vary [WRH⁺00]. The interviewees, and the project of which the research process were actively followed were based on convenience sampling, i.e., through recommendation. The projects that were surveyed were selected through theoretical sampling. All the cases were exclusively IT companies, and companies that incorporate an IT unit were not included. Thus, as open source is a strategy that can be adopted by the entire population, inclusion of such companies might have been beneficial.

Maturation

Maturation can act as an threat to the internal validity in relation with the surveyed project web sites. They might have modified their web sites as time has passed by, as well as what documentation that is presented, and what tools that are used. Through examining the dates of when the activity began in the tools, this threat has been negated as much as is possible. Additionally, as these projects all were released quite recently, the results can still be considered useful.

11.2.4 Threats to Construct Validity

Construct validity concerns generalizing the result to the concept or theory. This is dealt with through using multiple sources of evidence, corroborating the evidence. However, it could have been even better negated through performing additional interviews.

11.2.5 Threats to External Validity

Threats to external validity are conditions that limit the ability to generalize the results [WRH⁺00]. The major threat related to this project relates to whether provision of generalized guidelines is possible when it comes to open source, and whether the guidelines that are presented indeed are valid for a representative community. This threat is negated as much as possible through examining a wide variety of literature, and through corroboration of the literature findings with data gathered through interviews the findings from the surveyed web sits. The way the guidelines are presented, should thus be highly generalizable. Still, they might not represent the entire truth. Each company will have to assess these guidelines in relation with their company-specific situation, and consider which ones are applicable for their purposes.

11.3 Summary

This chapter has presented an evaluation of the project. It has described the findings related to each research question, and has presented the limitations and validity threats that have been identified in the project. An answer was provided to all the research questions based on the research approach taken. While there are some validity threats that might influence the theory that has been presented, the author believes that the initial problem description has been covered thoroughly.

Chapter 12

Conclusions and Future Work

This chapter presents the conclusions of the project, and provides suggestions for further work.

12.1 Conclusions

The purpose of this project was to describe the processes of commercial open source initiatives with a focus on the early stages of the release process. This included identifying the rationale behind commercial open source initiatives, and what companies can do to increase their chance of getting a successful project. The project was approached qualitatively, performing a literature survey, four interviews, and surveying the web sites of six commercial open source initiatives. With a basis in these sources, data was corroborated and general theoretical guidelines were created describing important aspects that should be considered before, during, and after the release of a commercial open source initiative.

Thus, the contributions of this thesis can be divided into two parts; the literature study, and the provision of general guidelines that companies may follow when considering the adoption of an open source strategy, i.e., to initiate an open source project. The author wishes to express his gratitude toward the entire open source community, and hopes that this thesis might act as an inspirational source for further research, and for increased adoption of open source strategies.

12.1.1 Literature Study

The first contribution of this thesis is the literature study. This literature study provides a broad description of the open source phenomenon, including its history and background, characteristics of open source, the concept of open source licensing, and motivations for commercial adoption of open source strategies. This description provides background information, and can be used as a source to gain general knowledge of the open source phenomenon. Knowledge of what open source entails is of importance when adopting an open source strategy. This literature study can also act as an introduction to open source for those seeking to learn more about what comprises the open source phenomenon.

12.1.2 General Guidelines

The second contribution provided in this thesis has been to establish important aspects and considerations to be made when releasing an open source project. These have been assembled as general guidelines, with a division into three parts: General knowledge of open source, preparations for the release, and processes that should be followed after the release.

General Knowledge of Open Source

When considering the adoption of an open source strategy, it is important to have a general understanding on the open source phenomenon. The most important knowledge that should be assembled has been described in Part II of this report, and have been listed in table 12.1:

General Knowledge	Purpose
What it is	<ul style="list-style-type: none"> -Knowledge of the history and background of open source. -Knowledge of what is meant by open source. -Knowledge of what the open source definition entails. -Knowledge of the open source community, motivations, organizational structures, and roles. -Knowledge of the open source development process.
Licensing	<ul style="list-style-type: none"> -Knowledge of the common properties found in open source licenses. -Knowledge of what the most important open source licenses entail.
Commercial Adoption	<ul style="list-style-type: none"> -Knowledge of how open source can be used by commercial actors. -Knowledge of how releasing software as open source can benefit the company. -Knowledge of business models that can be adopted.

Table 12.1: Guidelines related to general knowledge of open source

12.1.3 Preparations for the Release

There are several preparations that should be made before the release of an open source projects. These preparations are listed in Table 12.2:

12.1.4 After the project has been released

When the project has been released, promotional activities should be performed, and the company must make sure to maintain an active project, good leadership, and good interaction with the community, as listed in Table 12.3:

Preparations	Purpose
Identify the Rationale	The rationale behind the adoption of an open source strategy should be identified
Project Suitability	<ul style="list-style-type: none"> -Choose a suitable name. -Make sure the code works and contains basic functionality. -Estimate whether the software will meet a demand. -Estimate whether the project has potential for success. -Make sure that the architecture is fitting for open source.
Infrastructure	<ul style="list-style-type: none"> -Know the tools that can be used in open source projects. -Know of the benefits these tools can provide. -Select tools based on what is required, and what seems the most viable choices. -Decide whether a canned hosting solution should be used, and which tools to implement manually,
Web Site	<ul style="list-style-type: none"> -Provide a good design and layout. -Separate the structure, the presentation, and the dynamic content of the web site. -Make sure that it is presented so that search engines will rank it as high as possible. -Make sure the web site clearly presents the project. -Describe why the project should be used and contributed to. -Describe the features of the project. -Describe clearly that the project is open source, and which licenses are used. -Provide a description of the development status. -Provide feature requests. -Provide all documentation at the web site. -Provide links to all the tools that are used.

Table 12.2: Guidelines related to preparations for the release of open source software

12.2 Suggestions for Future Work

This research comprises a theoretical approach to establish important aspects and considerations to be made when releasing an open source project. While these guidelines should be able to act as an aid to increase the chance of achieving a successful projects, there is a need for further evaluation and refinement. The following suggestions for future work and argumentation of why such studies might be necessary, can act as an inspirational source for further research.

12.2.1 Quantitative Research

There have been a lack of quantitative data presented in this thesis. By performing a large-scale quantitative survey on commercial open source specifically designed to provide further evidence and

After the release	Actions
Promotional Activities	<ul style="list-style-type: none"> -Participate at developer gatherings. -Announce the project at related web sites. -Issue a press release, write articles for publishing in media of interest. -Actively engage users and communities, especially those that are likely to be of interest. -Issue newsletters if a customer base or community has been established. -If the resources are available, paid advertisements in targeted media can be viable. -Be aware of the external promotional effect that might occur, and encourage it.
Community Relations	<ul style="list-style-type: none"> -Make sure that the project comes forth as being active and healthy. -Maintain good relations with the community, taking their suggestions and comments into consideration. -Make sure that credit is provided to everyone that contributes.

Table 12.3: Guidelines related to the period after the release of open source software

verification of the guidelines, the findings might be further validated, refined, extended, and weak points might be identified. Quantitative research on unsuccessful open source initiatives might also be an idea for future research, as there exist few such studies. It was also noticed that only 50 percent of the examined open source projects provided public access to the source code repository. Although only six web sites were studied, this might indicate a trend in commercial open source initiatives that opposes the traditional view on the tools publicly used in open source projects. Performing a quantitative study investigating the presence of tools available in commercial open source projects is thus suggested.

12.2.2 Case Study

Case studies can be performed following companies applying the guidelines closely. This will help to further substantiate the evidence, to provide verification, and to further refine and extend the guidelines. Some of the guidelines provided might be difficult to measure per se, and the findings should be measured against a baseline where a comparable project has been performed that did not follow some, or most of these guidelines. There should also be performed further evaluation of what factors that might negatively impact the success of an open source project. There have, not surprisingly, been found little research on failed open source projects. Thus, qualitative interviews targeting companies that have been unsuccessful in their open source strategies are suggested as strategy to extend the guidelines.

Part V

Appendices

Appendix A

Interview Guide

This Appendix contains the introductory letter that were sent to the interview respondents, and the questions that were used to guide the interview¹.

A.1 Hensikt med intervjuet

Hensikten med dette intervjuet er å undersøke hvilke strategier bedrifter benytter seg av i forbindelse med åpen kildekode, med et særlig fokus på bedrifter som har startet sine egne åpen kildekode-prosjekter. Det er ønskelig å få informasjon om hvilke prosesser bedriften har fulgt for å tiltrekke seg utviklere og brukere, samt hvilken effekt de valgte strategier har hatt. Det vil også interessant å finne ut hvordan bedriften tilrettelegger og kommuniserer med de eksterne utviklere og brukerne, og om spesifikke tiltak utføres for å beholde disse.

A.2 Intervjuets kontekst

Jeg (Tor Erik) er masterstudent ved gruppen for systemutvikling ved IDI, NTNU, og disse intervjuene gjennomføres som en del av min masteroppgave. Masteroppgaven er gjennomført i forbindelse med ITEA COSI prosjektet ², der NTNU deltar som forskningspartner. COSI-prosjektet fokuserer på hvordan man kan forbedre bruk av åpen kildekode i industriell sammenheng. Hovedveiledere for prosjektet er Professor Reidar Conradi og Dr. Carl-Fredrik Sørensen.

A.3 Gjennomføring og behandling av informasjon

Intervjuene vil bli tatt opp med båndopptaker, slik at jeg kan forsikre meg om at ingen informasjon går til spille. Disse opptakene vil *ikke* bli frigjort til andre og vil behandles konfidensielt. Resultater

¹The introductory letter and the interview questions are available in Norwegian only.

²<http://www.itea-cosi.org>

fra intervjuene vil bli brukt i min masteroppgave, men vil om ønskelig kodifiseres slik at resultater hverken kan spores til respondent eller respondentens bedrift.

A.4 Annet

Dersom det er noen spørsmål, uklarheter e.l. til intervjuets gjennomføring er det bare å kontakte meg på email:(torerei@stud.ntnu.no) eller tlf: [Tatt bort]. Ellers står dere selvfølgelig fritt til å be om klarifikasjon/spesifisering av spørsmålene mine, eller til å la være å svare på enkelte spørsmål.

Mvh Tor Erik Eide

A.5 Intervju-spørsmål

A.5.1 Introduksjons-spørsmål

- Kan du fortelle om din bakgrunn?
 - Utdannelse og arbeidserfaring?
 - Erfaring med OS?
 - Rolle i firmaet/OS-prosjektet?
- Om firmaets OS-produkt
 - Når ble det bestemt at prosjektet skulle slippes som OS?
 - Hvorfor open source?
 - Hva tilbyr produktet fremfor konkurrenter?
 - Hvordan gikk dere fram i valg av lisens til prosjektet?
 - Hvor modent var produktet da det ble sluppet?
 - Ble elementer som ryddig kode, modularitet vektlagt?

A.5.2 Tiltrekke community

- Har dere foretatt aktive handlinger(markedsføring) med den hensikt å tiltrekke community?
- Hvis ja:
 - Når begynte dere med markedsføring (mens under intern utvikling? etter slipp?)
 - Hva har dere gjort for å skape bevissthet/awareness om at prosjektet eksisterer?
 - Hva har dere gjort for å skape interesse rundt prosjektet? (Tiltrekke community)
 - Har dere rettet markedsføring mot spesielle grupper(f.eks tidligere kunder), eller har det vært en mer generell markedsføring?
- Spesifisering av tiltak:

- Har dere presentert prosjektet på Workshops e.l?
- Har dere skapt publisitet om prosjektet (på forhånd og etter slipp) gjennom aktivitet i OS-miljø, blogging, posting i forumer/maillinglister, newsletters?
- Har dere publisert prosjektet i eksterne portaler som SourceForge, eller Freshmeat?
- Har dere sendt pressemelding, fått postet om prosjektet i nettaviser e.l?
- Har dere betalt for annonse noe sted, evt. i ikke-digitalt media?
- Har dere utvekslet bannere, publisering med andre firmaer, partnere?
- Har dere jobbet aktivt for å oppnå høyt prioriterte hits i søkemotorer?
- Hvis nei:
 - Kan dere begrunne hvorfor dette ikke er gjort?
 - Har dere lyktes å bygge et community rundt prosjektet likevel?
 - Hvordan har community fått kjennskap til og interesse for prosjektet?
- Hvilken effekt har de forskjellige markedsførings-tiltakene hatt?
 - Vil du si at strategien generelt sett har vært vellykket?
 - Hvordan får folk flest kjennskap til prosjektet?
 - Er det noe dere ville gjort annerledes idag, nå som dere har fått mer erfaring?
 - Hva ville dere *ikke* gjort annerledes?
 - Hva tror dere var den viktigste årsakene til at dere tiltrakk dere et community?
 - Hvor lang tid tok det før ting skjedde? I.e. mye aktivitet og evt. bidrag fra community?
 - Har communityt selv vært en viktig ressurs med tanke på å spre ordet/tiltrekke flere brukere/utviklere?

A.5.3 Prosesser rundt slipp av open source

- Hvordan gikk dere fram i valg av teknisk infrastruktur?
- Hvilken innvirkning har valg av teknisk infrastruktur hatt på prosjektets suksess?
 - Hvilken innflytelse har valg av portalløsning (Egen, SourceForge) hatt?
 - Hvilken innvirkning har verktøy tilgjengelig på hjemmesiden hatt for prosjektet?
 - Hadde dere valgt en annen løsning om dere skulle startet med prosjektet idag?
- Hvor viktig er presentasjonen av prosjektet med tanke på å tiltrekke seg utviklere og brukere?
 - Hvor viktig er design og layout?
 - Hvor viktig er dokumentasjon? (Bruker og utvikler)
- Hvordan presenterte dere prosjektet når dere slapp det som åpen kildekode?
 - La dere spesielt vekt på layout, brukervennlighet og design av hjemmesiden?

- Hva tilbød dere av verktøy, infrastruktur for kommunikasjon?
- Hva tilbød dere av dokumentasjon?
- Forklarte hjemmesiden tydelig hvordan man kunne bidra til prosjektet?
- Forklarte hjemmesiden tydelig hva dere ønsket av bidrag?
- Hva har forandret seg siden den gang, hvordan har hjemmesiden/portalen utviklet seg?

A.5.4 Interaksjon med community

- Hvordan var forholdet til communityet i startfasen av prosjektet?
 - Kommuniserte dere aktivt med community helt fra starten av? Hvor viktig er dette?
 - I hvilke(t) media foregår hovedtyngden av kommunikasjon. Mailingliste/Forum?
 - Kan kommunikasjon/interaksjon med community bidra til å tiltrekke brukere og utviklere?
 - Har dere møtt utfordringer basert på at dere er et kommersielt firma? Hvordan har isåfall disse blitt taklet?
- Hva har dere gjort/gjør dere for å tilrettelegge for community?
 - Tilbyr dere oppdatert dokumentasjon på hjemmesiden?
 - Er dere flinke til å oppdatere hjemmesiden?
 - Er dere flinke til å delta aktivt kommunikasjonsmessig, være tilgjengelige?
 - Gjennomfører dere noen spesielle tiltak for å bygge community-”følelsen”, i.e. IRL møter?
 - Benytter dere irc, andre instant messenger medier? Hvordan. Logges?
 - Har dere veikart for videre utvikling? Hvor viktig er det å tilby dette?
 - Holdes alle relevant kommunikasjon offentlig og tilgjengelig for alle?
 - Hvordan tas avgjørelser i prosjektet?
 - Har kundekrav ledet til konflikter med community?
 - Har dere intern mailing-liste/begrenset forum. Begrenset til hvem?
 - Betaler dere noen utviklere utenfor firmaet? Hvem skriver dokumentasjon?
 - Gir dere synlig credit til de som bidrar? Er dette viktig for community og for å opprettholde interesse?
 - Har dere spesifisert utviklingsprosess og/eller releaseprosess? Hvordan styres dette?

Appendix B

Interviewees

This Appendix briefly describes the companies that were interviewed, the interviewees, and the open source products that were focused on in the interviews.

B.1 Bekk Consulting

Bekk Consulting¹, is a Norwegian company delivering technology solutions and consultancy services to large organizations. Bekk employees spend much time to create useful tools to aid the consultant services, some of which are made open source. The representatives of Bekk Consulting was Stefan Landro and Christian Schwarz. Stefan Landro is the leader of the open source group in Bekk, while Christian Schwarz is one of the founders of the open source project, CubicTest².

The interview focused on MiddleGen³, which is an open source project released at SourceForge in October 2001. MiddleGen is a general-purpose, database-driven engine for code-generation, that has acquired much popularity with more than 200.000 downloads. CubicTest, an Eclipse plugin that aims to make the process of creating automatic acceptance tests for AJAX web applications easier, was also briefly discussed.

B.2 eZ Systems

eZ Systems⁴ was founded in 1999 and is today an international company with offices in Norway, Ukraine, France, Canada and Germany⁵. They provide commercial support, training, workshops, certification, consulting and integration services. They also apply the dual licensing model for their flagship product, eZ Publish, thus selling proprietary licenses in addition to providing the software under GNU GPL. The interviewee representing eZ Systems was Vidar Langseid, Head of Development.

¹<http://www.bekk.no>

²<http://boss.bekk.no/display/BOSS/CubicTest>

³<http://sourceforge.net/projects/middlegen/>

⁴<http://www.ez.no>

⁵http://ez.no/company/about_ez

The project that was the focus of the interview was eZ Publish, which is an enterprise content management system first released at SourceForge in October 2000 ⁶. The project quickly gained much attention, and it was downloaded more than 4000 times the first month after release. Subsequently, the project was moved off SourceForge the 23rd of November 2000.

B.3 Linpro

Linpro⁷ was formed in 1995, being the first Norwegian company having Linux as a dedicated business area. They are today one of the leading Norwegian companies in the area of providing products and services in relation with general open source software and Linux. The interviewee representing Linpro was Thomas Malt, substitute and forthcoming head of development.

The project that was the main focus of the interview, is known as Varnish ⁸. Varnish is an HTTP-accelerator that according to Linpro⁹ provides ten to twenty times the performance of it's competitors. The project was initiated after a request and sponsorship by VG multimedia, and publicly released in September 2006. Linpro provides commercial installation, integration, maintenance, and support services in relation with Varnish.

B.4 Trolltech

Trolltech¹⁰ was founded in 1994, and uses dual licensing as a business model, providing both commercial and open source licenses. In addition to selling commercial licenses, the company also offers support, training and consultancy services. The interviewee was Knut Yrvin, community manager of Trolltech.

Qt is a cross-platform application development framework that have made Trolltech known worldwide. It is, e.g., used in software such as KDE, Opera, and Google Earth. Qt was first released as a Graphical User Interface toolkit in 1996. At this time, it was available for free, but distribution of modifications was not allowed [Hec00]. The project was used in the KDE project, causing a stir within the community because of the license, and the creation of competing projects ¹¹. In 2000, Qt was released under a "true" open source license, the QPL and later that year it was also available under the GPL ending the conflict with the FSF proponents.

⁶<http://sourceforge.net/projects/ezpublish/>

⁷<http://www.linpro.no>

⁸<http://varinsh.projects.linopro.no>

⁹http://www.linpro.no/no/produkter_og_loesninger/aapen_kildekode_loesninger_prosjekter/varnish

¹⁰<http://www.trolltech.com>

¹¹<http://www.kde.org/whatiskde/qt.php>

Appendix C

Project Web Sites

C.1 Assessment Questions

- What tools are available from the web site?
- What documentation is available from the web site?
- How does the web page present issues related to development status and feature requests?

C.2 Openbravo

Openbravo¹ is an open source ERP business solution for small and medium sized companies. It was released as open source at SourceForge March 2006 under the Openbravo Public License².

C.3 TYPOLight

TYPOLight webCMS³ is a CMS that specializes in accessibility and uses XHTML and CSS to generate W3C/WAI compliant pages. It was released as open source at SourceForge February 2006 under GPL.

C.4 netWiser

Netwiser⁴ is a platform for network software development, that includes a graphical user interface, a network simulator, and a portable C++ framework for creating network-intensive applications. It was released as open source May 2006 under GPL.

¹[urlhttp://www.openbravo.com](http://www.openbravo.com)

²Openbravo Public License is almost similar to MPL.

³<http://www.typolight.org>

⁴http://bitwiserlabs.com/index.php?option=com_content&task=section&id=5&Itemid=36

C.5 OpenEMM

OpenEMM⁵ is an industrial-strength enterprise software for professional e-mail newsletters and e-mail marketing. It was released as open source June 2006 under MPL.

C.6 Visual WebGUI

Visual WebGUI⁶ is a framework for creating AJAX-enabled applications in .NET. It was released as open source April 2006 under the GPL, but has now changed to LGPL.

C.7 Webload

Webload⁷ is a commercial-grade load testing solution licensed under GPL. It was released as open source February 2007.

⁵<http://http://www.openemm.org/>

⁶<http://www.visualwebgui.com/>

⁷<http://www.weblload.org/>

Appendix D

Keywatch Web Site

```
* Home
* Download
* Mailing List
* Documentation
* Contact

* Join the Keywatch Community

You are here: Keywatch > Home
* User Name
* _____
* Password
* _____
* Log in

* Forgot your password?
* New user?

Keywatch

- an open source, flexible, osgi-based monitoring system

Keywatch is a free, open source monitoring system that provides the
adaptability of a modern monitoring system without being complex or
hard to configure.

Keywatch runs on most platforms. The server runs Java OSGi and the
client technology is AJAX, requiring only a standard web browser.

By offering Keywatch under the Apache 2 lisenece, we hope to build a
community that will contribute functionality for Keywatch to support
common monitoring challenges and a lot of specific ones, making
Keywatch an attractive alternative to expensive and less flexible
systems.
Valuechain

Main features

* Keywatch supports the full monitoring value chain.
* High flexibility - write plugins in Java and extend collection any
step of the value chain.
* Distributed monitoring - Agent based collection system using
XMLRPC
* Automate manual monitoring tasks - Agents able to run arbitrary
scripts to collect information
* Monitor HW, SW, services, SLA or other objects

See the features site for more detailed featurelisting.

News

About Keymind | About Portal | Privacy Policy | Contact Us © 2007,
Norges Tekniske Naturvitenskaplige Universitet
```

Figure D.1: Keywatch web site in Lynx

Home Download Mailing List Documentation Bugtracker Contact


Keywatch

- an open source, flexible, osgi-based monitoring system

Keywatch is a free, open source monitoring system that provides the adaptability of a modern monitoring system without being complex or hard to configure.

Keywatch runs on most platforms. The server runs Java OSGi and the client technology is AJAX, requiring only a standard web browser.

By offering Keywatch under the [Apache 2 license](#), we hope to build a community that will contribute functionality for Keywatch to support common monitoring challenges and a lot of specific ones, making Keywatch an attractive alternative to expensive and less flexible systems.



Main features

- Keywatch supports the full monitoring value chain.
- High flexibility - write plugins in Java and extend collection any step of the value chain.
- Distributed monitoring - Agent based collection system using XMLRPC
- Automate manual monitoring tasks - Agents able to run arbitrary scripts to collect information
- Monitor HW, SW, services, SLA or other objects

See the [features site](#) for more detailed featurelisting.

News

2007-03-27 Test

Testnyhet

[About Keymind](#) | [About Portal](#) | [Privacy Policy](#) | [Contact Us](#)

User Name

Password

Remember Me:

[Forgot your password?](#)
[New user?](#)

Figure D.2: Layout and design of the Keywatch web site

Appendix E

Glossary

API Application Programming Interface

ARPANET Advanced Research Projects Agency Network

ASF Apache Software Foundation

CSS Cascading Style Sheet

COSI Co-development using inner & Open source in Software Intensive products

CPL Common Public License

CVS Concurrent Version System

DARPA Defense Advanced Research Projects Agency

EPL Eclipse Public License

FLOSS Free/Libre Open Source Software

FOSS Free/Open Source Software

FSF Free Software Foundation

GPL General Public License

HTML HyperText Markup Language

IM Instant Messenger

IRC Internet Relay Chat

IPR Intellectual Property Rights

IT Information Technology

LGPL Lesser General Public License

NCSA National Center for Supercomputing Applications

MPL Mozilla Public License

OSI Open Source Initiative

OSS Open Source Software

OTS Off The Shelf (Components)

RFC Request For Comments

SVN Subversion

XML eXtended Markup Language

Bibliography

- [ABNR05] Paul Adams, Cornelia Boldyreff, David Nutter, and Stephen Rank. Adaptive Reuse of Libre Software Systems for Supporting On-line Collaboration. In *Proceedings of the fifth workshop on Open source software engineering*, pages 1–4, St. Louis, Missouri, 2005. ACM SIGSOFT.
- [App07] Net Applications. Browser Market Share for May, 2007. <http://marketshare.hitslink.com/report.aspx?qprid=0>, 2007. [Online, Accessed 2007-06-17].
- [Aro02] Lars Aronsson. Operation of a Large Scale General Purpose Wiki Website. In *Proceedings of the 6th International ICCC/IFIP Conference on Electronic Publishing*, pages 27–37, Karlovy Vary, Czech Republic, 2002. Verlag für Wissenschaft und Forschung.
- [BC03] Carliss Y. Baldwin and Kim B. Clark. Does Code Architecture Mitigate Free Riding in the Open Source Development Model? <http://opensource.mit.edu/papers/baldwin.pdf>, 2003. Online, Accessed: 2007-05-16].
- [BCK05] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison Wesley, 2005.
- [Ben05] Lorenzo Benussi. Analyzing the technological History of the Open Source Phenomenon. Stories from the Software Evolution (Working Paper). <http://opensource.mit.edu/papers/benussi.pdf>, 2005. Presented at O’Reilly European Open Source Conference 2006 (EuroOSCON 2006) [Online, Accessed: 2007-03-06].
- [Bev04] Nigel Bevan. Usability Issues in Web Site Design. Technical report, Serco Usability Services, 2004.
- [BI06] Judith Bar-Ilan. Web links and Search Engine Ranking: Google and the Query ”Jew”. *Journal of the American Society for Information Science and Technology*, 57:1581–1589, 2006.
- [BL01] Magnus Bergquist and Jan Ljungberg. The power of gifts: Organizing social relationships in open source communities. *Information Systems Journal*, 11:205–220, 2001.
- [BM01] Shirley A. Becker and Florence E. Mottay. A Global Perspective on Web Site Usability. *IEEE Software*, 18:54–61, 2001.
- [BR03] Andrea Bonaccorsi and Cristina Rossi. Why Open Source software can succeed. *Research Policy*, 32:1243–1258, 2003.

- [BR06a] Andrea Bonaccorsi and Cristina Rossi. Comparing Motivations of Individual Programmers and Firms to Take Part in the Open Source Movement: From Community to Business. *Knowledge, Technology, and Policy*, 18:40–64, 2006.
- [Bär06b] Matthias Bärwolff. Tight prior open source equilibrium: The rise of open source as a source of economic welfare. *First Monday*, 11, 2006.
- [BSKK02] Brian Butler, Lee Sproull, Sara Kiesler, and Robert Kraut. Community Effort in Online Groups: Who Does the Work and Why? <http://opensource.mit.edu/papers/butler.pdf>, 2002. In: Weisband, S.A.L., (Ed.) *Leadership at a Distance*. Erlbaum, In Press. Butler, B.S. 2001. Membership Size, Communication Activity, and Sustainability: A ResourceBased Model of Online Social Structures. *Information Systems Research*. 12, 346-362 [Online, Accessed: 2007-03-15].
- [CAH03] Kevin Crowston, Hala Annabi, and James Howison. Defining Open Source Software Project Success. In *Proceedings of International Conference on Information Systems (ICIS 2003)*. Association for Information Systems, 2003.
- [CLM⁺07] Weibing Chen, Jingyue Li, Jingquian Ma, Reidar Conradi, Junzhong Ji, and Chunlian Liu. A Survey of Software Development with Open Source Components in Chinese Software Industry. <http://www.idi.ntnu.no/grupper/su/publ/li/icsp07-oss-china.pdf>, 2007. Accepted by International Conference on Software Process 2007 (ICSP 2007) [Online, Accessed: 2007-03-29].
- [CLW⁺07] Kevin Croston, Qing Li, Kangning Wei, U. Yeliz Eseryel, and James Howison. Self-organization of teams for free/libre open source software development. *Information and software technology*, 49:564–575, 2007.
- [CTW98] Andrea Chávez, Catherine Tornabene, and Gio Wiederhold. Software Component Licensing: A Primer. *IEEE Software*, 15:47–53, 1998.
- [CW98] Reidar Conradi and Bernhard Westfechtel. Version Models for Software Configuration Management. *ACM Computing Surveys*, 30:232–282, 1998.
- [DGMN02] Jamie Dinkelacker, Pankaj K. Garg, Rob Miller, and Dean Nelson. Progressive open source. In *Proceedings of the 24th International Conference on Software Engineering*, pages 177–184, New York, NY, USA, 2002. ACM Press.
- [DL03] Benoit Demil and Xavier Lecocq. Neither Market nor Hierarchy or Network: The emerging Bazaar Governance. <http://opensource.mit.edu/papers/demillecocq.pdf>, 2003. Used at the: "XIV ième Conférence Internationale de Management Stratégique", [Online, Accessed 2007-03-04].
- [dM03] Joseph dal Molin. Open Source Software In Canada - A Collaborative FactFinding Study. Technical report, e-cology corporation, 2003.
- [DM05] Linus Dahlander and Mats. G. Magnusson. Relationships Between Open Source Software Companies and Communities: Observations from Nordic Firms. *Research Policy*, 4:481–493, 2005.

-
- [Eis89] Kathleen M. Eisenhardt. Building Theories from Case Study Research. *The Academy of Management review*, 14:532–550, 1989.
- [Eis91] Kathleen M. Eisenhardt. Better Stories and Better Constructs: The Case for Rigor and Comparative Logic. *The Academy of Management review*, 16:620–627, 1991.
- [ES06] Tor Erik Eide and Per Kristian Schanke. Going Open: Guidelines for Commercial Actors to Release Software as Open Source. <http://www.idi.ntnu.no/grupper/su/fordypningsprosjekt-2006/eide-schanke-fordyp06.pdf>, 2006. Depth Project at the Norwegian University of Science and Technology, [Online, Accessed 2007-05-29].
- [FF02] Joseph Feller and Brian Fitzgerald. *Understanding Open Source Software Development*. Addison-Wesley Professional, 2002.
- [Fit04] Brian Fitzgerald. A Critical Look at Open Source. *IEEE Computer*, 37:92–94, 2004.
- [Fit06] Brian Fitzgerald. The transformation of open source software. *MIS Quarterly*, 30, 2006.
- [Fog05] Karl Fogel. *Producing Open Source Software*. O’Reilly Media, 2005.
- [Fou07a] The Eclipse Foundation. About the Eclipse Foundation. <http://www.eclipse.org/org>, 2007. [Online, Accessed: 2007-05-14].
- [Fou07b] The Eclipse Foundation. Eclipse Public License - Frequently Asked Questions. <http://www.eclipse.org/legal/eplfaq.php>, 2007. [Online, Accessed: 2007-05-14].
- [GA04] Cristina Gacek and Budi Arief. The Many Meanings of Open Source. *IEEE Software*, 21:34–40, 2004.
- [GG05] Ron Goldman and Richard P. Gabriel. *Innovation Happens Elsewhere*. Elsevier Inc, 2005.
- [GGKR02] Rishab A. Ghosh, Ruediger Glott, Bernhard Kriger, and Gregorio Robles. Free/Libre and Open Source Software: Survey and Study(Part 4: Survey of Developers). Technical report, UNU-MERIT, 2002.
- [Gho06] Rishab Aiyer Ghosh. Study on the: Economic impact of open source software on innovation and competitiveness of the Information and Communication Technologies(ICT) in the EU. Technical report, UNU-MERIT, 2006.
- [Goo07] Google. Webmaster Guidelines. <http://www.google.com/support/webmasters/bin/answer.py?answer=35769>, 2007. [Online, Accessed: 2007-06-16].
- [GS67] Barney G. Glaser and Anselm Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine Transaction, 1967.
- [Hec00] Frank Hecker. Setting up shop: The Business of Open Source Software. *IEEE Software*, 16, 2000.
- [HO02] Alexander Hars and Shaosong Ou. Working for Free? Motivations for Participating in Open-Source Projects. *International Journal of Electronic Commerce*, 6, 2002.

- [HR06] Øyvind Hauge and Andreas Røsdal. A Survey of Industrial Involvement in Open Source. Master's thesis, Norwegian University of Science and Technology, 2006.
- [JF00] Brian Fitzgerald Joseph Feller. A Framework Analysis of the Open Source Software Development Paradigm. In *Proceedings of the twenty first international conference on Information systems*, pages 58–69, Atlanta, GA, USA, 2000. Association for Information Systems.
- [Kha00] Asif Khalak. Economic Model for Impact of Open Source Software. <http://opensource.mit.edu/papers/osseconomics.pdf>, 2000. [Online, Accessed 2007-06-20].
- [Kin01] Shannon Kinnard. *Marketing with E mail: A Spam Free Guide to Increasing Sales, Building Loyalty, and Increasing Awareness*. Maximum Press, 2001.
- [KM01] Bruce Kogut and Anca Metiu. Open-Source Software development and Distributed Innovation. *Oxford Review of Economic Policy*, 17:248–262, 2001.
- [Kri02] Sandeep Krishnamurthy. Cave or Community? An Empirical Examination of 100 Mature Open Source Projects. <http://www.opensource.mit.edu/papers/krishnamurthy.pdf>, 2002. [Online, accessed 2007-03-22].
- [Kri03] Sandeep Krishnamurthy. An Analysis of Open Source Business Models. <http://faculty.washington.edu/sandeep/d/bazaar.pdf>, 2003. [Online, Accessed 2007-03-28].
- [Kri05] Sandeep Krishnamurthy. The Launching of Mozilla Firefox - A Case Study in Community-Led Marketing (Working Paper). <http://opensource.mit.edu/papers/sandeep2.pdf>, 2005. [Online, Accessed: 2007-05-14].
- [KW01] Gerhard Kleining and Harald Witt. Discovery as Basic Methodology of Qualitative and Quantitative Research. *Forum: Qualitative Social Research*, 2:1–24, 2001.
- [LC06] Rui Pedro Lourenco and João Paulo Costa. Discursive e-Democracy Support. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, pages 69–79. IEEE Computer Society, 2006.
- [LCS+05] Jingyue Li, Reidar Conradi, Odd Petter N. Slyngstad, Christian Bunse, Umir Khan, Marco Torchiano, and Maurizio Morisio. An Empirical Study on Off-the-Shelf Component Usage in Industrial Projects. In *Proceedings of the 6th International Conference on Product Focused Software Process Improvement (PROFES'2005)*, pages 54–68, Oulu, Finland, 2005. Springer.
- [Les02] Donald M. Leslie. Using Javadoc and XML to produce API reference documentation. In *Proceedings of the 20th annual international conference on Computer documentation*, pages 104–109, 2002.
- [Lin04] Juho Lindman. Effects of Open Source Software on the Business Patterns of Software Industry. Master's thesis, Helsinki School of Economics, 2004.

-
- [LPT06] Josh Lerner, Parag A. Pathak, and Jean Tirole. The Dynamics of Open-Source Contributions. *The American Economic Review*, 96:114–118, 2006.
- [LT02] Josh Lerner and Jean Tirole. The Simple Economics of Open Source. *Journal of Industrial Economics*, 50:197–234, 2002.
- [LT05a] Josh Lerner and Jean Tirole. The Economics of Technology Sharing: Open Source and Beyond. *The Journal of Economic Perspectives*, 19:99–120, 2005.
- [LT05b] Josh Lerner and Jean Tirole. The Scope of Open Source Licensing. *The Journal of Law, Economics & Organization*, 21:20–56, 2005.
- [LW03] Karim R. Lakhani and Robert G. Wolf. Why Hackers Do What They Do: Understanding Motivation Effort in Free/Open Source Software Projects. Technical report, MIT Sloan School of Management, 2003.
- [Mal01] Kirsti Malterud. Qualitative research: standards, challenges, and guidelines. *The Lancet*, 358:483–488, 2001.
- [MFH02] Audris Mockus, Roy T. Fielding, and James D. Herbsleb. Two case studies of Open Source Software development: Apache and Mozilla. *ACM Transactions*, 11:309–346, 2002.
- [MFT] Gregory Madey, Vincent Freeh, and Renee Tynan. Modeling the F/OSS Community: A Quantitative Investigation. In Book: Free Open Source Software Development.
- [Mic05] Martin Michlmayr. Software Process Maturity and the Success of Free Software Projects. <http://opensource.mit.edu/papers/michlmayr1.pdf>, 2005. [Online, Accessed: 2007-04-30].
- [Mor05] Håvard Mork. Leadership in Hybrid Commercial-Open Source Software Development. Master’s thesis, Norwegian University of Science and Technology, NTNU, 2005.
- [Moz98] Mozilla.org. Mozilla & Netscape Public Licenses. <http://www.mozilla.org/MPL/MPL-1.1.html>, 1998. [Online, Accessed 2007-04-23].
- [moz07a] mozilla.org. About Mozilla. <http://www.mozilla.org/about/>, 2007. [Online, Accessed 2007-04-30].
- [moz07b] mozilla.org. About the Mozilla Foundation, 2007. [Online, Accessed 2007-04-30].
- [MS02] David C. Mowery and Timothy Simcoe. Is the Internet a U.S Invention? An Economic and Technological History of Computer Networking. *Research Policy*, 31:1369–1387, 2002.
- [Nel65] Theodor Holm Nelson. Complex Information processing: a File Structure for the complex, the changing and the Indeterminate. In *Proceedings of the 1965 20th national conference*, pages 84–100, Cleveland, Ohio, United States, 1965. ACM Press.
- [NT03] David M. Nichols and Michael B. Twidale. The Usability of Open Source Software. *First Monday*, 8, 2003.

- [oA06] Det Kongelige Fornyings og Administrasjonsdepartement. St.prp.nr.1 (2006-2007) for budsjettåret 2007. http://www.statsbudsjettet.dep.no/upload/Statsbudsjett_2007/dokumenter/pdf/fagdep/fad.pdf, 2006. [Online, Accessed 2007-03-12].
- [O'R99] Tim O'Reilly. Lessons from Open-Source Software Development. *Communications of the ACM*, 42:32–37, 1999.
- [Ray02] Eric Steven Raymond. *The Cathedral and the Bazaar*. O'Reilly Media, 2002.
- [RdMF02] Christian Robottom Reis and Renata Pontin de Mattos Fortes. An Overview of the Software Engineering Process and Tools in the Mozilla Project. In *Proceedings of the Open Source Software Development Workshop*, pages 155–175, Newcastle upon Tyne, UK, 2002. Newcastle University E-prints.
- [RE04] Michel Ruffin and Christof Ebert. Using Open Source Software in Product Development: A Primer. *IEEE Software*, 21:82–86, 2004.
- [Ric91] Richard Stallman. The GNU General Public License (GPL) v.2. <http://www.opensource.org/licenses/gpl-license.php>, 1991. [Online, Accessed 2007-04-23].
- [Rob06] Jennifer Niederst Robbins. *Web Design in a Nutshell, 3rd Edition*. O'Reilly Media, 2006.
- [Sam06] Pamela Samuelson. IBM's pragmatic embrace of open source. *Communications of the ACM*, 49:21–25, 2006.
- [SC05] Nicolás Serrano and Ismael Ciordia. Bugzilla, ITracker, and other Bug Trackers. *IEEE Software*, 22:11–13, 2005.
- [Sca02] Walt Scacchi. Understanding the Requirements for Developing Open Source Software Systems. In *Software, IEE Proceedings*, pages 24–39. IET, 2002.
- [Sca04] Walt Scacchi. Free and Open Source Development Practices in the Game Community. *IEEE Software*, 21:59–66, 2004.
- [Sca07] Walt Scacchi. *Free/Open Source Software Development: Recent Research Results and Methods*. Elsevier, 2007.
- [Sch01] Patrice-Emmanuel Schmitz. Study into the use of Open Source Software in the Public Sector. Technical report, European Commission, DG Enterprise, 2001.
- [SFT07] Param Vir Singh, Ming Fan, and Yong Tan. An Empirical Investigation of Code Contribution, Communication Participation, and Release Strategy in Open Source Software Development: A Conditional Hazard Model Approach. http://opensource.mit.edu/papers/singh_fan_tan.pdf, 2007. [Online, Accessed 2007-03-18].
- [SMHM99] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. *ACM SIGIR Forum*, 33:6–12, 1999.
- [Stü06] Matthias Stürmer. Open Source Community Building. Master's thesis, University of Bern, 2006.

-
- [Sta99] Richard Stallman. GNU Lesser General Public License. <http://www.opensource.org/licenses/lgpl-license.php>, 1999. [Online, Accessed 2007-04-23].
- [SWJS01] Amanda Spink, Dietmar Wolfram, Major B.J. Jansen, and Tefko Saracevic. Searching the Web: The Public and Their Queries. *Journal of the American Society for Information Science and Technology*, 52:226–234, 2001.
- [The04] The Apache Software Foundation. Apache License, Version 2.0. <http://www.apache.org/licenses/LICENSE-2.0>, 2004. [Online, Accessed 2007-04-23].
- [Thu03] Shari Thurow. *Search Engine Visibility*. New Riders, 2003.
- [Tuo01] Ilkka Tuomi. Internet, Innovation and Open Source: Actors in the Network. *First Monday*, 6:1–1, 2001.
- [Uni99] University of California. The bsd license. <http://www.opensource.org/licenses/bsd-license.php>, 1999. [Online, Accessed 2007-04-23].
- [vHvK03] Eric von Hippel and Georg von Krogh. Open Source Software and the “Private-Collective” Innovation Model: Issues for Organization Science. *Organization Science*, 14:213–223, 2003.
- [vKSL03] Georg von Krogh, Sebastian Spaeth, and Karim R. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32:1217–1241, 2003.
- [VS02] Dapzury Valenzuela and Pallavi Shrivastava. Interview as a Method for Qualitative Research. <http://www.public.asu.edu/~kroel/www500/Interview%20Fri.pdf>, 2002. [Online, Accessed: 2007-02-23].
- [Web03] Steven Weber. Open Source Software in Developing Economies. Technical report, Social Science Research Council, 2003.
- [Wes03] Joel West. How open is open enough? Melding proprietary and open source platform strategies. *Research Policy*, 33:1259–1285, 2003.
- [Whe05] David A. Wheeler. Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers! http://www.dwheeler.com/oss_fs_why.html, 2005. [Online, Accessed 2007-06-20].
- [WM05] Joel West and Siobhán Mahony. Contrasting Community Building in Sponsored and Community Founded Open Source Projects. In *Proceedings of the 38th Hawaii International Conference on System Sciences*, pages 196–206, Washington, DC, USA, 2005. IEEE Computer Society.
- [WRH⁺00] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in Software Engineering*. Kluwer Academic Publishers, 2000.
- [WW01] Huaiqing Wang and Chen Wang. Open Source Software Adaptation: A Status Report. *IEEE Software*, 18:90–95, 2001.

- [Yin03] Robert K. Yin. *Case Study Research. Design and Methods - Third Edition*. Sage Publications, 2003.
- [YYSI00] Yutaka Yamauchi, Makoto Yokozawa, Takeshi Shinohara, and Toru Ishida. Collaboration with Lean Media: How Open-Source Software Succeeds. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 329–338, Philadelphia, Pennsylvania, United States, 2000. ACM Press.
- [ZD05] Jin Zhang and Alexandra Dimitroff. The impact of metadata implementation on webpage visibility in search engine results(Part II). *Information Processing and Management*, 41:691–715, 2005.