



OPEN SOURCE SOFTWARE MANAGEMENT FRAMEWORK

*Nina Helander, Timo Aaltonen, Teemu Mikkonen, Ville Oksanen,
Mikko Puhakka, Marko Seppänen, Tere Vadén, Niklas Vainio*

Distribution

eBRC

Tampere University of Technology and University of Tampere

Published by

Tampere University of Technology (TUT) and University of Tampere (UTA)

Printed edition

ISSN 1459-0158

ISBN 978-952-15-1855-3 (TUT)

978-951-44-7110-0 (UTA)

Electronic edition in <http://www.ebrc.fi>

ISSN 1459-0166

ISBN 978-952-15-1856-0 (TUT)

978-951-44-7111-7 (UTA)

Printed by Cityoffset Oy, Tampere 2007

Table of Contents

FOREWORD	2
1 PART I: INTRODUCTION	5
1.1 OVERVIEW TO OSS BUSINESS AND ITS DEVELOPMENT	5
1.2 BASIS FOR THE RESEARCH: DIFFERENT OSS USER ROLES	6
1.3 STRUCTURE OF THE REPORT	8
2 PART II: GUIDELINES FOR SUCCESSFUL COMMUNITY PARTICIPATION	9
2.1 FACTS ABOUT THE CASE COMMUNITIES	9
2.1.1 DEBIAN	9
2.1.2 GNOME	10
2.1.3 ECLIPSE	11
2.1.4 MYSQL	12
2.2 TOOLS FOR RECOGNIZING SUSTAINABILITY RISKS	13
2.3 CONCLUSION: TYPOLOGY OF OS COMMUNITIES	14
3 PART III: TOWARDS SUCCESSFUL OPEN SOURCE PROJECT EVALUATION	19
3.1 INTRODUCTION TO EVALUATION OF OSS PROJECTS FROM BUSINESS PERSPECTIVE	19
3.2 DIFFERENT EVALUATION TOOLS FOR OSS PROJECTS	20
3.2.1 OPTAROS' MODEL	20
3.2.2 OPEN SOURCE MATURITY MODEL (OSMM)	21
3.2.3 QUALIFICATION AND SELECTION OF OPEN SOURCE SOFTWARE (QSOS)	23
3.2.4 BUSINESS READINESS RATING (BRR)	26
3.2.5 COMPARISON AND CONCLUSION OF THE SELECTED MODELS	30
3.3 EMPIRICAL TEST: EVALUATING GNOME WITH BRR MODEL	31
3.3.1 CHOOSING AN EXAMPLE SOFTWARE	31
3.3.2 APPLYING BUSINESS READINESS RATING	31
3.4 LESSONS LEARNT	34
4 PART IV: CONCLUSIONS	38
4.1 MANAGEMENT GUIDELINES FOR DIFFERENT OSS USER TYPES	38
4.2 EVALUATION OF THE RESULTS AND INSIGHTS FOR FURTHER OSS STUDIES	42
REFERENCES	44
APPENDIX 1: OPEN SOURCE BUSINESS REFLECTIONS	45
APPENDIX 2: INTRODUCTION TO LICENSE CHECKER	50
APPENDIX 3: VALUE NETWORK ANALYSIS OF DEBIAN AND ECLIPSE	68
APPENDIX 4: OSSI RESEARCH IN A NUTSHELL	87

Foreword

Without no doubt it can be stated that OSS is a multifaceted phenomenon that from the companies' perspectives affects the ways of doing R&D, HR, marketing, sales, communications, legal operations, etc. It is inherently such a multidimensional phenomenon that it would be hard to understand it, if we would look it only from one perspective. Instead, we need to have a multidisciplinary approach.

These kinds of fundamental thoughts were expressed by Dr. Ari Jaaksi from Nokia Multimedia and Professors Tommi Mikkonen and Saku Mäkinen from Tampere University of Technology, when they started to write the first draft of the OSSI research plan. As acting the ideological fathers of this project, we would like to express our deepest gratitude to these gentlemen. Without them the project would have not been kicked off.

In order to build a strong multidisciplinary research project we need an idea, but also a network. The network would not have been created without director Petri Räsänen from COSS, director Marko Seppä from eBRC and Professor Juha Laine from Helsinki University of Technology, who gave their valuable time to build and develop the network, in which researchers from Helsinki University of Technology, Tampere University of Technology, University of Tampere and Helsinki School of Economics have been able to cooperate successfully. We wish to most warmly thank these gentlemen for their networking skills, but also for their supervision throughout the project.

We also want to express our sincerest gratitude to Tekes and Verso technology program, which have made the research project possible. Especially we want to thank Keith Bonnici and Matti Sihto.

We have had the privilege to work with a number of great industrial partners during the OSSI project. We are extremely grateful to Nokia Multimedia, IBM, Nokia Siemens Networks, ABB, F-Secure, Plenware Group, PricewaterhouseCoopers, Teknologiakeskus Hermia, TietoEnator, SUN Microsystems and WM-data, with whom we have had the chance to study as inspiring phenomenon as OSS is.

Additionally, we would like to give special thanks to all of our collaboration partners. Especially we would like to thank Stephen Walli, for his valuable comments during our research work and his invaluable comments on this report. We would also like to thank Marjut Anderson from COSS and Hanna Martin-Vahvanen and Maria Antikainen for their valuable efforts in OSSI project. We would also like to thank the COSI research project for fruitful co-operation, as well as all other OSS researchers in Finland.

OSSI as a research project has lasted two years, from 1.7.2005 to 30.6.2007, but we as a group of researchers will continue our research work together in future projects. We strongly believe that OSS is a basis for new businesses and business models, offering innovative possibilities for different kind of actors: service providers, software developers, system integrators, end user organizations and individuals, etc. However, as OSS challenges many of the old and familiar ways companies to operate, we need

further understanding on OSS, both on a strategic and operational level. That is why we believe that successful utilization of OSS requires new kind of thinking and innovative management tools and models. To contribute on our part to this development, we will continue, enthusiastically, our studies on OSS.

At this point of our research journey, it is however, time to sum up the results we have achieved so far. In this report that serves as the final report of OSSI research project, we will summarize the main results of OSSI research project. However, in the earlier research reports of OSSI, 1) Essays on OSS Practices and Sustainability, 2) Empirical Insights on Open Source Software Business and 3) Multidisciplinary Views to Open Source Software Business, important and complementary views and results are also brought up. Throughout the project we have also actively published in academic journals and edited books (especially to mention the recently published Handbook on Open Source Software Research). At the end of this report, a list of these publications with a short description on the issues covered in them is offered.

This report concentrates on presenting the tools and models that together form the OSS management framework. The results are written by keeping mind the needs of companies, as OSSI project has foremost been an industry driven research guiding the researchers to work together to bring multidisciplinary, but yet unified view for companies looking to utilize OSS.

We would like to point out that our research results are based on analysis of four Open Source communities, Eclipse, GNOME, Debian and MySQL. Although these four case communities represent different types of Open Source communities, they still are just four of the over 100 000 Open Source projects. Thus, the generalizability of our research results should be carefully kept in mind when applying the results in practice.

This report comprises four sections, the introduction part, the community part, the OSS evaluation part and finally, the conclusion. In addition to this main content of the report, there are four appendices, which bring up complementary views on OSS, in the end of the report.

In the introduction of the report, a short overview to OSS business development is presented. We will also present the base for our analysis, a typology of different OSS utilization roles that is further used in our analysis and in the development of the OSS management tools. We would like to point out that the typology serves well for analytical purposes, but in practice the differences between different OSS utilization roles are not so clear.

The Part II of the report concentrates on the four case communities of the research. We present the basic facts and characteristics of these communities as well as some results of OSSI surveys. Finally, the Part II presents ways of distinguishing between different types of communities, guidelines to identify potential community risks and best practices for interaction with different community types. The sustainability analysis tool is presented in this part of the report.

The Part III deals with the issue of OSS evaluation. The questions that are crucial for evaluation are presented, before we go through four existing evaluation models. On

the basis of analysis of these models, we will present an improved evaluation model, which offers a more comprehensive view to OSS evaluation.

In the fourth part of the report, the conclusions are presented. In the conclusions, check-lists and guidelines relevant to each OSS utilization role are presented. These include considerations from technological, sociological, business and legal perspectives.

In the end of the report are appendices that include complementary issues to the main results, as they bring forward Open Source business reflections, introduce the license checker tool developed as one important OSS management tool during the OSSI project, present a comparative value network analysis of two case communities and lastly, summarize the publications of OSSI project by themes.

We hope that all organizations that seek to benefit from OSS find this report useful!

OSSI Research group

1 PART I: INTRODUCTION

1.1 Overview to OSS business and its development

The development of Open Source Software (OSS) business can be captured in the sentence “From Free Software to Open Source and Commercial Open Source”. We summarize the history of OSS business in three eras. The summary is, however, simplistic on purpose, and the borders between the eras are not sharp. For instance, the practice of sharing software and working on it collaboratively is as old as software itself, predating the Internet by decades. Correspondingly, co-operatively written and shared software has been a part of commercially marketed products at least since the 80's. The eras are as following:

Era I: 1985 - Free Software and Richard Stallman

In 1985 in order to promote the idea of freedom, Stallman founded The Free Software Foundation (FSF). For the goals of the Foundation he outlined the four freedoms of Free Software that should be an industry practice:

- 1) *The freedom to run the program as you wish;*
- 2) *The freedom to study the source code and to change it to do what you wish;*
- 3) *The freedom to make copies and distribute them to others; and*
- 4) *The freedom to publish or more generally, distribute modified versions.*

In 1989 FSF introduced the GPL (General Public License) to be used in distributing software under terms meeting the ideas of the four freedoms of software. Two years later, in 1991, GPLv2 was introduced. The GPL continues to be the most popular license type used both for free software and open source (FOSS). In short, from Stallman's viewpoint the idea of free or open software is an issue of ethics and ethical behavior, not technical superiority or business interests.

Era II: 1998 - Open Source Software and Eric Raymond

Eric Raymond felt early on that the most radical thing about GNU/Linux was not the fact that it was the first free operating system, but in that by creating the kernel Linus Torvalds had invented a totally new way of developing software by making use of thousands of volunteer developers collaborating over Internet in a distributed “organization” towards a common goal. This new “Bazaar-style” development methodology is, in Raymond's view, a better, more efficient way than the traditional hierarchical and controlled way. He crystallized the benefits in the now famous slogan “given enough eyeballs, all bugs are shallow.” The direction of Raymond's quote is often taken on the back end, i.e. once a bug is noted, the number of people involved can quickly make short work of the bug. However, the actual strength of Linus's law can be understood on the front end. Any form of software inspection finds more bugs than testing. Hence, code being submitted on the reflectors is inspected by far more people in a well run project than would normally happen in a traditional top down project (See e.g. Conradi et al. 1999). By 1997 he had written and released the classic essay “The Cathedral and the Bazaar” that highlights the key issues in this new approach. In February 1998 Raymond founded the Open Source Initiative, and quickly got support during the year as, e.g., IBM, Sun Microsystems, Oracle, Informix and Corel

announced initiatives to support Open Source. This meant Open Source and some ideas (not all!) of Free Software were making their way into big businesses and their software practices.

Era III: 2005 - Commercial Open Source Software

After several years of steady growth in popularity of open source, and it having started to challenge incumbents in many fronts, in 2005 a new term appeared “Commercial Open Source Software”, used by, e.g., Microsoft. With the rising popularity of open source, that has somewhat challenging, or even disruptive, business-models for many incumbent software companies, this new term was introduced to describe, e.g., mixed source products where part of the code is open and part proprietary, making it possible to offer own products under the same license fee based model as before, while getting the benefits of open source code without having to pay a license fee on that. This of course is quite far from the ethical ideas originated by Stallman, but it is easy to see why businesses are trying this.

We have come from an idealistic goal of doing things in the ethical way into a world where the models, such as GPL license, are used not to give or promote freedom of developers but rather the business interests of both small and large companies, sometimes at the expense of developers’ freedom. Nevertheless, the role of companies is growing in the Open Source field, and the next question could perhaps be if Open Source becomes business as usual.

One thing is for sure; Open Source is here to stay. It has gained a strong foothold in software business, and nowadays other businesses are also keen to find what kinds of new ideas and ways of operation it may offer. Best practices of open source communities to govern and to develop are increasingly studied and transferred into other areas of business. Still, there exist many issues yet to be solved within software business. Governance methods and communication structures typically used in open source do not fit well with the typical organisation structure of a firm. To merge these two aspects needs a lot of effort and understanding of the best of both halves. Answers are needed to questions like: How the sustainability of Open Source procurement can be guaranteed?, How can we utilise open source in our business?, and How to manage the use of OSS and the interaction with the OSS communities? In OSSI research project, these issues were in foci.

1.2 Basis for the research: different OSS user roles

As OSS changes many of the basic rules of software development and business, companies that use open source software in their business need a comprehensive view on how to deal with the phenomena, not just one narrow view from a specific scientific discipline. In OSSI, this multifaceted phenomenon has been studied from the viewpoints of technology, sociology, law and business.

In Figure 1, below, these aspects are illustrated. From the technological point of view, the main questions have been the structure and quality of OSS code in comparison to code produced by other, more traditional, views. This point of view has been closely connected with the sociological one, the main question of which has been the

motivations and socio-cultural backgrounds of the developers. The changing legal framework and licensing practices have been the focus of the perspective from the point of view of the law. All of these have informed and been informed by the business perspective, where the main question has been the developing landscape of building business on OSS. The research question that ties all these strands together is: *How can the interaction between commercial companies and OSS communities best be managed?*

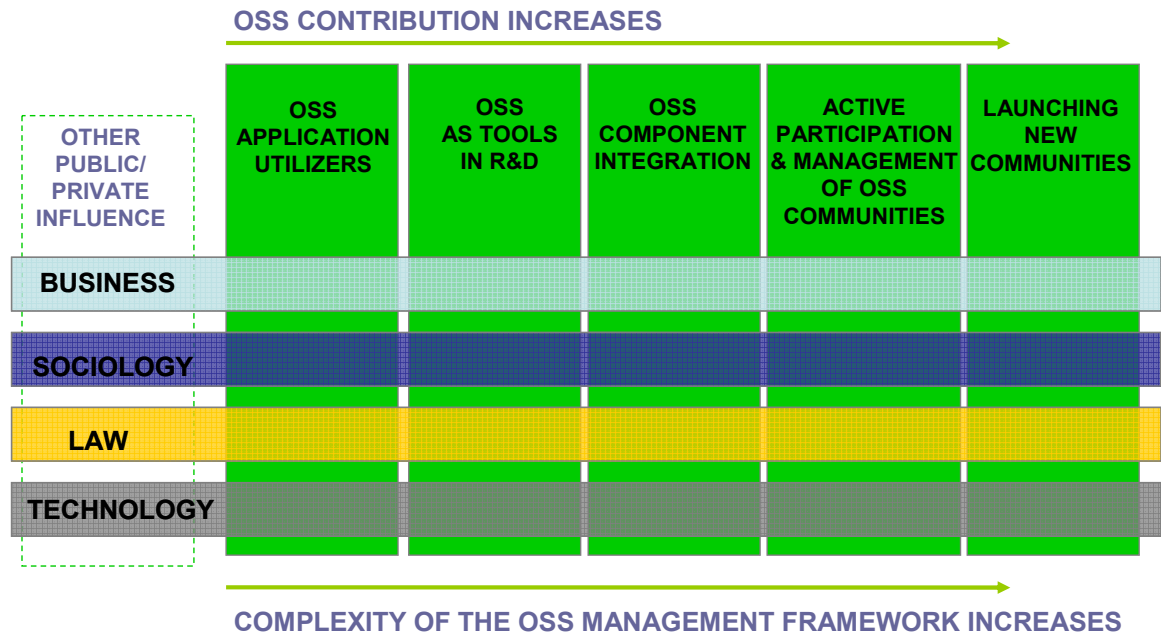


Figure 1. Different OSS user types have different needs for OSS management framework

In addition to these aspects, an important viewpoint is the different roles that companies in participating in the OSS may play. These five stages acted as the starting point for our analysis in OSS project, including stages from utilizing OSS applications to launching new communities, which are situated along a continuous scale of ever increasing intensity of OSS involvement. The research assumption is that as the involvement intensifies, also the task of managing that involvement gets more complex and crucial.

The two first roles, OSS application utilization and OSS as tools in R&D, are the least demanding. However, already here it is important to evaluate the quality of the OSS applications to be used, and to assess the longevity of the communities behind the applications. When we move to OSS component integration, the role of the community interaction increases. A company doing the interaction will have to be involved with several communities, and the success of one or several of these will be decisive for the longevity of the integrated software. The sustainability and productivity of these communities will have to be evaluated, and a strategy for being a good “open source citizen” developed for mutually beneficial community interaction. The role of active management and participation in OSS communities is needed, for instance, when OSS software is a crucial element in a product or service provided by the company. The company will have to take an active, maybe even leading role in the life of the community. This might mean taking part in the decision making, contributing to the organisation of the community, if such exists, and hiring

developers in/from the community or other means of remuneration. Consequently, more refined and complex tasks of community evaluation have to be undertaken, and an explicit and detailed management framework developed and implemented. A vivid understanding of the differences between types of OSS communities and the historical development they go through becomes necessary.

Finally, a company might decide that it needs to launch a new community by itself. Here being a good OSS citizen is not enough. One has to show consistent leadership and demonstrate staying in power in the sifting market. The competition for talented developers is a competition for mind-share, so a factor of coolness and promises of great things to come have to be given – and also kept. Issues of sustainability and community type become paramount.

1.3 Structure of the report

This report comprises of four sections, the introduction part, the community part, the OSS evaluation part and finally, the conclusion. In addition to this basic content of the report, there are four appendices, which bring up complementary views on OSS, in the end of the report.

In the introduction of the report, a short overview to OSS business development was presented. The illustration of the different OS utilization roles was presented, as these act as the bases for the development of the OSS management tools for companies.

Part II concentrates on the communities chosen to be empirically studied in the OSSI project. We present the main facts and characteristics of these communities, as well as some results of OSSI surveys. Finally, part II presents ways of distinguishing between different types of communities, guidelines to identify potential community risks and best practices for interaction with different community types.

Part III deals with the issue of OSS evaluation. The questions that are crucial for evaluation are presented, before we go through four existing evaluation models, Optaros' Enterprise Readiness (ER) model, Open Source Maturity Model (OSMM) by B. Golden, a model for Qualification and Selection of Open Source Software (QSOS), and finally, Business Readiness Rating (BRR). On the basis of these models, part III also presents the outlines of an evaluation model that we feel to be an improvement on the existing ones.

Finally, in the conclusion part the main results are presented.

2 PART II: GUIDELINES FOR SUCCESSFUL COMMUNITY PARTICIPATION

2.1 Facts about the case communities

The communities in our study were Debian, Gnome, Eclipse and MySQL. The four communities were chosen to represent different types of OS projects. In the following, central facts about each of these communities are presented. The text bases partly on the results of our survey of these case communities. The survey was carried out in 2005-2006 and the main results of the survey were presented in the OSSI report "Empirical insights on Open Source Software".

2.1.1 Debian

Debian GNU/Linux is one of the oldest Linux-based distributions still in existence. The project started in August 1993 making it the oldest of the four. Debian has been and still is a project based on volunteer work. The Debian community doesn't produce software in the narrow sense of the word; instead it focuses on packaging existing free/open source software to integrate it in the Debian operating system. At the moment, Debian consists of 19 000 packages, which are maintained by approximately 2000 maintainers. The distribution is popular as a server operating system but it is being used also in workstations and embedded devices. Ubuntu Linux is a variant (fork) of Debian that is currently gaining popularity.

Debian developers would like to see companies to "Certify products on Debian"; e.g., by stickers saying "Works on/with Debian"

The social organisation of Debian is built around the technological architecture (packages). Each package has a maintainer (sometimes a few maintainers) that has the primary responsibility for the package. Common packaging rules, guidelines and principles have been compiled into guides such as the Debian Policy Manual. But not only technical decisions are being regulated like this. The most important values of Debian have been codified into the Debian Social Contract. This document, together with its appendix Debian Free Software Guidelines, defined the value basis and goals of the project. The Social Contract emphasises freedom of the software and promises to keep Debian fully free, transparent and to give back to the free software community. Software packages in Debian have different licences, but all of them need to meet the requirements of Debian Free Software Guidelines to be included in the distribution. Software that is freely distributable does not fully meet the criteria (e.g. if the software may not be used for commercial purposes), and is sometimes included in the unofficial "non-free" section.

Debian has also a formal organisation that is defined in the Debian Constitution. The project has a Project Leader elected every year by maintainers that have received the official status of Debian Developer. The role of the project leader is to coordinate and to represent the project, and the Leader may appoint delegates to coordinate some

tasks. Debian Developers also have the right to propose General Resolutions or Constitutional Amendments that are voted upon. For example the status of software and documentation that doesn't meet the Free Software Guidelines criteria has been the topic of a heated discussion.

Debian has formalized its *modus operandi* with a written Debian Social Contract and the Debian Constitution"

In our survey, the Debian Project was also the largest community. The project maintains official mailing lists, technical infrastructure and conferences for communication and coordination between developers. In 2005 there were 965 (2007: 1013 <http://www.us.debian.org/vote/2007/vote_001_quorum.log>) developers with voting rights in the yearly elections (504 did vote) and the developer database contained 1411 names (22.11. '05). The majority of developers in our survey were highly educated males with degrees, but there was a significant amount of high school student also (22%).

Debian is famous for being one of the most freedom oriented, politically aware and volunteer-centered FOSS communities. Its longevity and robustness are probably results of the well maintained community focus. Volunteers have been active because their freedom and the freedom of the code has been guaranteed. Consequently, the recent introduction of monetary rewards for developers has not been smooth. A project called Dunc-Tank was launched in 2006 to provide funding for developers working on Debian. This has allegedly prompted some developers not involved in Dunc-Tank to reduce their effort or even abandon work on Debian, citing the creation of a "two-class" system as the reason for their disenchantment. However, a clear majority (95%) of Debian developers see company participation in OS development as a good thing (for details, see Mikkonen, Vaden & Vainio 2007).

2.1.2 GNOME

The Gnome (originally from the acronym GNU Network Object Model Environment) project is an effort to build a free software desktop environment. It was launched in 1997 by the GNU project and is licensed under LGPL for its libraries and GPL for the actual parts of the program.

The Gnome project is loosely organized and the discussion chiefly occurs on a number of public mailing lists. GNOME is an umbrella for software which is used in conjunction of an operating system like Linux and Solaris. It is an organized community with ca. 1000 members who are working in usability, accessibility and QA teams. The GNOME release team has defined new releases to occur every six months. The project is given structure by the Gnome Foundation that has a Board of Directors as well as an Advisory Board. The advisory board consists of members from companies and other entities that support Gnome, including, e.g., ACCESS, Canonical, Debian Project, Free Software Foundation, Hewlett-Packard, IBM, Imendio, Intel, Nokia, Novell, OLPC, OpenedHand, Red Hat, Software Freedom Law Center, Sun Microsystems.

The GNOME Foundation oversees the development of the project & brings community & companies together.

GNOME is ten years old with relatively young developers. In our study (see Mikkonen, Vadén & Vainio 2007) the mean age of the GNOME developers was 27. GNOME developers are mainly men and they are quite highly educated. Most of the GNOME developers in our survey had Bachelors degrees but there were significant amount of Masters, too. The developers in GNOME community are mostly volunteers, but there are also some developers who get a salary for developing GNOME. The GNOME community is closer to Debian than for example Eclipse or MySQL with it's relatively poorly salaried, young volunteers.

GNOME developers suggest to companies: Do quality assurance work!

2.1.3 ECLIPSE

Eclipse is a platform independent software framework for delivering so called rich-client applications. It was founded 2001 by a consortium pf companies including, e.g., Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft and Webgain. Later the consortium has grown to over 80 members. Eclipse platform became open source when IBM released it 2004.

Eclipse is an umbrella project composed of many different software projects. In its internet sites it describes itself as "*an open source community whose projects are focused on providing a vendor-neutral open development platform and application frameworks for building software*". There is a non-profit foundation behind the Eclipse community which is "*formed to advance the creation, evolution, promotion, and support of the Eclipse Platform and to cultivate both an open source community and an ecosystem of complementary products, capabilities, and services*". Eclipse hosts 9 major open source projects with over 50 subprojects and it is estimated to have about 500 developers. Consequently, it is difficult to estimate how many developers are in the overall community.

In our survey most of the developers in Eclipse were professional and usually got paid for their work on Eclipse. They are also mostly middle-aged men (mean age is 38) and have better incomes then the developers of GNOME or Debian. It is still difficult to say how these characteristics describe Eclipse community generally, because of the fragmented structure. Eclipse is a community of the communities and it might be helpful to research these communities also separately.

Eclipse has a detailed and still developer friendly IP-policy

Corporate legal departments are typically wary of relative chaotic nature of IP-management found from most of the small and midsize open source projects. However, due to its corporate roots, Eclipse has given special attention to this area. The project shows how a very detailed and safe IP-policy can be established without alienating the developers. Special attention has been given to make the language in legal sections as understandable as possible for non-lawyers. Additional agreements are used to ensure that all contributions are “made by the rightful copyright holder and under the Eclipse Public License (EPL).”

Eclipse developers expect from companies: Explore business models that exploit EPL code.

2.1.4 MySQL

MySQL is a multithreaded, multi-user, SQL Database Management System. MySQL is available both as free (GPL) and as proprietary software. The company MySQL AB develops and maintains the system, selling support and service contracts, as well as proprietary-licensed copies of MySQL. Both volunteers and employees of MySQL (the company) participate in development.

There were already in about 1979 first ideas and code conceived for MySQL, but the development of MySQL itself started in 1995 with a first release at the end of 1996. MySQL has over 300 employees in over 25 countries and is one of the largest open source companies worldwide. Together with Linux, Apache and PHP/Perl/Python, MySQL forms one of the building blocks of the LAMP technology stack. The MySQL AB claims a user base of over 8 million MySQL installations worldwide, and over 50,000 product downloads in daily. In 2005 MySQL reached about 40 million euro revenues, and stated having reached profitability.

In our survey there were only 14 answers from MySQL, so the representativeness isn't very high. Still, if you look at the results of our survey and compare them to the knowledge which is available in internet, it is easy to find some characteristics of the MySQL community. In our survey almost all participants got some salary from their work related to MySQL. They were highly educated and mostly men. Almost all developers (80%) were working in MySQL AB.

Correspondingly, MySQL has centralized decision-making system. The community behind MySQL is small compared for example to Debian and this was one reason for the small amount of answers.

Widely recommended form of support to communities: Company donates hardware or other resources.

2.2 Tools for recognizing sustainability risks

As can be seen from the characterisations of the four communities, above, OSS communities do share some characteristics (male dominance, relatively high level of education), but are also different in important respects. For instance, the developers of Eclipse and MySQL are as a trend roughly ten years older than those of GNOME and Debian. Also the motivations for participating in OSS development are different (for details, see Mikkonen, Vainio & Vadén 2007). Consequently, we need tools to assess the risks that the communities face: in a well known way most of the OSS projects listed on sites like Sourceforge are either dead or “communities” of one. Recognising some of the bottle-necks of community growth and sustainability will help a long way in establishing fruitful co-operation.

Below, the evaluatory questions are grouped in four sets, cultural, social, legal and economic. Social sustainability of a community relies on the individual characteristics of its members, on its size and form, and the division of labor and power in the community. Cultural sustainability of a community is defined by its traditions and history that create and shape its social and ethical norms and practices. While social sustainability is a matter of interaction between individuals, cultural sustainability is something that is created during a longer time period as the community matures. The importance of legal risk management in the OSS world has risen sharply during the last decade. The economic significance of software has drawn also the attention of the legal community and as the result the risk of getting sued for patent or copyright infringement is today very real. Finally, economic sustainability is one matter in volunteer based communities, and quite another in communities led by strong companies. However, for both extremes the problem of resources is anything but solved, and different models are constantly evolving and experimented with.

Social

1. Are there more than 20 active developers?
2. Does the community have a trusted main developer?
3. Does the community have developers with high technical skills?
4. Is the project cool enough to attract new developers?

Cultural

1. Does the community have a charter that defines the common principles and goals?
2. Is the development process open and inclusive?
3. Does the community have members who participate for ideological reasons?
4. Does the community have members that work for pay?

Legal

1. Does the community have legal expertise?
2. Does the software use a major open source license?
3. Does the software handle legally risky topics (p2p, encryption etc.)?
4. Does the economic footprint of the community attract law suits?

Economic

1. Is the maintenance of technical infrastructure on a sustainable basis?
2. Is some of the development work funded by companies?
3. Are some companies dependent on the community?
4. Does the community have funding for conferences and workshops?

2.3 Conclusion: typology of OS communities

In our survey, all of the communities were positive towards company participation. Thus, the main question for company participation is not *if* it is desirable and beneficial, but rather *how*. We suggest that FOSS communities may be divided into different categories according to idealised types, and that answering to the "how" question of participation must be differentiated according to these (idealised) typologies.

Traditionally, OSS communities have been started as volunteer projects (e.g., GNU project, Linux kernel, Debian). The traditional picture of hacker culture (see, e.g., Raymod 1999, Levy 1984) as an informal self-organizing bazaar of having fun while programming has largely been based on volunteer communities like these. However, the traditional picture has recently changed considerably with more and more companies participating in OSS communities either by letting their employees work on OSS or by directly hiring developers working on OSS. Increasingly companies also initiate OSS communities either by releasing previously closed code or by directly engaging in OSS development from the start. Consequently, a continuum of communities from volunteer-based to company-based has appeared. Most generally, this shift can be observed on the level of the *ethos* of communities: the ideologically organized ways in which labor is understood, maintained and given meaning. The self-organizing volunteer way of "working for fun" has been dubbed "hacker ethics" by, e.g., Himanen (2001). Himanen wants to explicitly contrast hacker ethics with the

more well-known salary-based commercial ethics that prevails that prevails in modern corporations, where a division and rationalization of labor takes place based on institutional rules and hierarchies (Himanen, 2001; see also Lash, 2002).

Consequently, the characterization of OSS communities to volunteer-based or company-based does not mean (mainly) the initiation of the project, but rather the basic ideological framework that motivates and structures the operations of the community. Typically, a company-based community has hierarchical structures, employs monetary rewards and divides labor on the basis of preset goals. In contrast, volunteer-based communities are self-organized, ground motivation on extra-monetary rewards and work on the basis of informal goal-setting (either anarchic, democratic or meritocratic). Typically, OSS communities today are a mix of the two extremes.

The work ethics of a community are closely tied to forms of decision making. Typically a self-organized community will favor decentralized decision making. One extreme is given by the decision on release dates in Debian: whenever the release is ready. In contrast, software development by and in a company will typically be centralized, with one source of authority deciding on, e.g., roadmaps and schedules. A middle ground between these two extremes is often sought by establishing a foundation or a similar organ that gives voice both to volunteers and the various institutions taking an interest in a given software development project. The foundation may guide development and structure schedules. Furthermore, communities may be classified on the basis of their age or maturity, size and the type of license in use:

- 1) Size of the community.** We assume that a larger community is always more sustainable but potentially increases problem complexity for company participation. The size of the community must also reach a certain minimum size in order to make the open source effect work.
- 2) Maturity of the community.** By maturity we mean the strength of the social and cultural ties, traditions and practices. A mature community is often old in age, and has developed common guidelines and best practices.
- 3) Communication and decision-making structures of the community.** Different systems of governance exist in free/open source software communities, including democracy, meritocracy and dictatorship. Here we look at how centralised communication is. This tells something about the governance structure, hierarchy and bottlenecks.
- 4) License.** The type of free/open source software license chosen by the community potentially affects who will participate in the community. We classify licenses based on how strong copyleft effect they have. GNU General Public License, for example, is a strong copyleft license, while Eclipse Public License gives more freedom, and licenses like the BSD license are not copyleft at all.

When we combine these four elements with the volunteer/company axis, differences between communities can be identified as can be seen in Table 1 (with examples).

Table 1. Community typology

		<i>Hybridity</i>		
		Volunteer	Mixed	Company
<i>Size</i>	Small	Wordpress		MySQL, Laika
	Medium	OpenBSD	Mozilla	OpenSolaris
	Large	Debian	Linux (kernel), GNOME	Eclipse
<i>Maturity</i>	Young	Gnash		Laika
	Developing	Wordpress	Mozilla	OpenSolaris, Darwin
	Established	GNU, Debian	Linux (kernel)	MySQL
<i>Decision-making</i>	Decentralized	Debian		Eclipse
	Balanced		Linux (kernel)	
	Centralized	GNU	Mozilla	MySQL
<i>License</i>	Non-copyleft	OpenBSD	Apache	
	Weak copyleft		Mozilla	Eclipse, OpenSolaris, Darwin
	Strong copyleft	GNU	Linux (kernel), GNOME	MySQL

In the classification above, we can see both differences and similarities between communities. Based on this analysis, some ideal types can be identified which characterise some of the most prominent differences between communities. Four ideal types could be identified:

- a) Centralized, company-driven, small community (e.g. MySQL)
- b) Large community, several companies, business work ethics (e.g. Eclipse)
- c) Large community, several companies, hacker background (e.g. Linux kernel)
- d) Volunteer, decentralized, large (e.g. Debian)

Correspondingly, different types of co-operation suit these types. Typically, small communities are more vulnerable. The risk of losing high-profile developers is considerable. On the other end, large communities often contain some inertia and may be susceptible to forks and internal disputes. From the perspective of sustainability, a large community that has also many participating companies is ideal. Diversity is the key to longevity in the open source ecosystem, as elsewhere.

a) Centralized, company-driven, small community

- do: direct co-operation with the company
- do: customization in co-operation with the company
- risk: sustainability dependent on single company

b) Large community, several companies, business work ethics

- do: involve own developers in the community
- do: collaboration with companies
- do: genuine contribution to community
- do: involvement in the decision making organs (e.g., Eclipse Foundation)
- don't: expect spontaneous development of code

c) Large community, several companies, hacker background

- do: involve own developers in the community
- do: quality contributions ("Show me the code!")
- do: involvement in the Open Source Development Labs
- do: good open source citizenship and sharing
- do: acknowledge community values
- don't try to push development without participating and contributing

d) Volunteer, decentralized, large

- do: support community (public acknowledgement)
- do: acknowledge community values
- do: be aware of licensing policies
- do: in case of a problem, do-it-yourself
- don't: use the software against the license terms
- risk: internal tensions
- risk: hard to keep deadlines

In Figure 2, this community typology and the position of the case communities are illustrated.

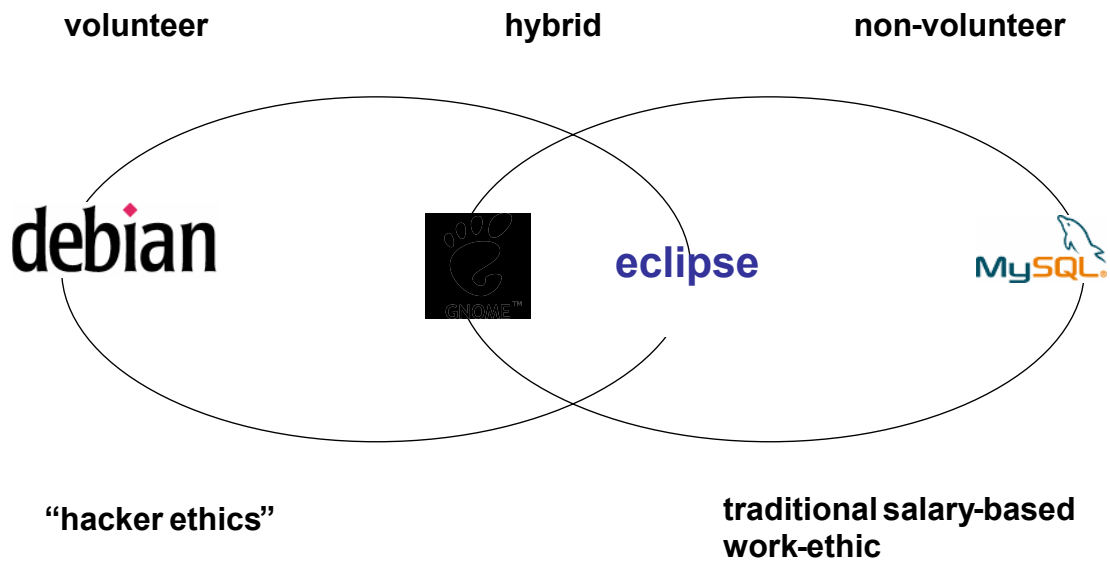


Figure 2. Community typology with regard to work ethics and the positions of the case communities

3 PART III: TOWARDS SUCCESSFUL OPEN SOURCE PROJECT EVALUATION

3.1 Introduction to evaluation of OSS projects from business perspective

Using open source software as a part of business has two sides, internal and external. Usually most of analysis focuses on external issues, for instance how to select the best piece of software or how to assess the viability of a particular community. However, internal issues may play vital role in succeeding implementation of open source.

Internal analysis should start with recognizing current and future needs. Questions that are useful in recognizing the reasons behind selecting open source software are, for example, the following:

- Analyzing time scale and urgency
 - How soon the output should be on market?
 - What is the overall life cycle of the output?
- Analyzing firms own resources and competences
 - What competences you need to a) select b) acquire c) maintain a software (this issue relates closely to outsourcing/purchasing)
 - How much resources you are able to invest for this issue?
- Analyzing the reasons to use open source software
 - Can you recognize your explicit and implicit motifs?
 - Why it is a strategic decision?
 - What are the main drivers?
 - What is the proposed use: are you going to use that particular piece of software in experimenting, piloting or production?
- Analyzing the status of relevant information
 - Do you know what you do not know?
- Analyzing the future
 - When the decisions are made, what consequences will follow?

An assessment task is about tradeoff between accuracy and time (i.e., money). Depending on answers on the questions above, one should make decisions what will be the needed level of information. An analytical and detailed approach may be too time- or resource-consuming when the software is just being experimented.

Another issue is the subject of the assessment. What is actually being assessed or evaluated? Is it a piece of software? It's source code? Or is it the quality of the source code? The project? What about the community producing and maintaining the source code? Is that community vital enough to ensure the participants of its future, and so on. Most of the typical evaluation tools for open source products or project are focused on assessing its completeness or maturity.

3.2 Different evaluation tools for OSS projects

In the following sub-chapters, we provide descriptions of four evaluation tool which are developed for assessing Open Source products or projects, namely Optaros' Enterprise Readiness (ER) model, Open Source Maturity Model (OSMM) by B. Golden, a model for Qualification and Selection of Open Source Software (QSOS), and finally, Business Readiness Rating (BRR). These models can be regarded as the "best-of-breed" according to our knowledge. The following presentations of evaluation models are primarily based on the core sources of each model including e.g. the model's website etc.

3.2.1 OPTAROS' Model

Optaros (2007) is an international consulting and systems integration firm that has created a catalog to provide a list of products best suited for today's enterprises. The catalog and its on-line version complemented with case studies and other information are available in <http://www.eosdirectory.com>. Only the products that match the enterprise benchmark in terms of functionality, community backing as well as maturity are listed. Technologies/projects are evaluated against four criteria:

- a. *Functionality* is compared with what is usually needed (e.g. in commercial products).
- b. *Community* demonstrates activity and support of the community behind the project.
- c. *Maturity* measures quality and robustness of a software product
- d. *Trend* indicates the expected future progress of the software product

Enterprise Readiness (ER) rating indicates how capable an open source software is to cope with the needs and requirements of midsize and large enterprises and organizations. ER-rating is indicated by one, two or three stars (Optaros' catalog does not list products that do not at least meet the one star level).

Table 2. Optaros ER ratings for Gnome, Debian, MySQL and Eclipse.

Product	Version	Description/ URL	License	Support	Function-ality	Comm-unity	Maturity	ER-Rating	Trend
Gnome	2.14	Graphical desktop environment for Linux http://www.gnome.org/	GPL	Community	✓✓✓✓	****	*****	◆◆◆	→
Debian GNU /Linux	3.1	Widely used Linux distribution http://www.debian.org	GPL	Community	✓✓✓✓	****	*****	◆◆	→
MySQL	5.0.22	Widely used open source relational database http://www.mysql.com/	GPL	Prof / Community	✓✓✓✓	****	*****	◆◆◆	↗
Eclipse	3.2	Leading Java IDE. The foundation was inherited from IBM VisualAge http://www.eclipse.org/	Eclipse Public License	Community	✓✓✓✓	****	*****	◆◆◆	↗

According to Optaros, for many applications, an open source product with a smaller functionality scope might be the better choice than a more complex one that does

more than what is needed. Moreover, in other situations, a simpler tool may be easier to integrate than a comprehensive one using another technology.

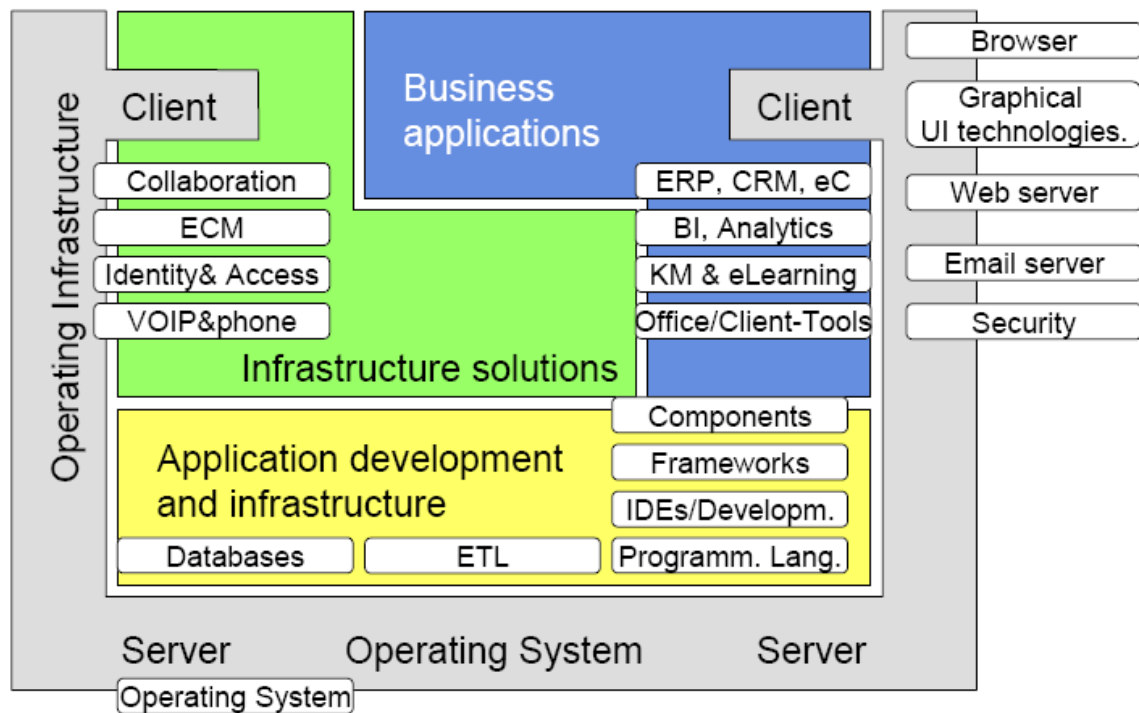


Figure 3. Four software categories and some of the covered subcategories in Optaros' assessments.

3.2.2 Open Source Maturity Model (OSMM)

The Open Source Maturity Model™ (OSMM) was created by Bernard Golden in 2004. The OSMM provides a framework to determine maturity level of an open source product. Its purpose is to enable a quick assessment of the maturity level of a given open source product. It offers great power to organizations evaluating the production readiness of an open source product, and to demonstrate its power a real assessment for JBOSS is performed.

The basic question is how to choose the best candidate? By using the OSMM, the products can be ranked according to their OSMM scores. Golden states that the model is designed to enable one or two people to spend no more than three to five days developing an overall maturity score for a product (i.e., to carry out a desk check).

The OSMM assesses a product's maturity in three phases:

- a. Assess each product element's maturity and assign a maturity score
- b. Define a weighting for each element based on the organization's requirements
- c. Calculate the product's overall maturity score

Table 3. Open Source Maturity Model with default weightings (OSMM, Golden 2004).

	Phase 1: Assess element maturity				Phase 2	Phase 3
	Define requirements	Locate resources	Assess element maturity	Assign element score	Assign weighting factor	Calculate product maturity score
Product software					4	
Support					2	
Documentation					1	
Training					1	
Product integrations					1	
Professional services					1	

In a typical maturity assessment, score scale is from 1 to 10. The template in Table X is available in <http://www.navicasoft.com/> as well as the example assessment for Drupal. Each step in the table is closely examined in the following.

Phase 1

- *Define organizational requirements* for a particular element. This is a key step to assess the usefulness of a product for a particular organization.
- *Locate resources*. Locating resources for an element is more challenging for open source products, but each chapter (in the book) offers a number of methods to identify that can assist an organization in implementing open source software.
- *Assess element maturity*. Determining where the element lies on the maturity continuum – from non-existent to production-ready – lets an organization determine how likely the product will be to satisfy its requirements.
- *Assign element maturity score*. Assignment provides the assessment of how well the product meets the organization’s requirements. This score documents the consensus of the organization. The process of determining the score requires the members of the assessment team to resolve differences in perception, make concrete the reasons for their judgment, and come to a common agreement about the product element. The maturity score serves as an input into improving the element’s maturity. Elements with low maturity score can be improved by the organization.

Phase 2

- *Apply product element weightings.* Weighting allows each element to reflect its importance to the overall maturity of the product. Default weightings in OSMM are shown in Table X above. These weightings may be needed to adjust based on the specific needs of the organization. The only limitation is that the sum of maturity weightings must be ten.

Phase 3

- *Calculate the product's overall maturity score.* After each element has been assessed and assigned a weighting factor, the overall product maturity score can be calculated. The elements scores are summed to give an overall maturity score on a scale of 1 to 100 that may be compared against recommended levels for different purposes, which vary according to whether an organization is an early adopter or a pragmatic user of IT. The following table lists recommended minimum OSMM scores.

Table 4. Recommend minimum OSMM scores
(<http://www.navicasoft.com/pages/osmmoverview.htm>)

	Type of User	
<i>Purpose of Use</i>	<i>Early Adopter</i>	<i>Pragmatist</i>
Experimentation	25	60
Pilot	40	60
Production	40	70

3.2.3 Qualification and Selection of Open Source Software (QSOS)

For a company, the selection to choose software as a component of its information system, whether this software is Open Source or commercially, rest on the analysis of the needs and constraints (technical, functional and strategic) and on the adequacy of the software to these needs and constraints. Atos Origin has conceived and formalized the QSOS method to ease this multi-faceted issue. They have made it available to all under the terms of GNU Free Documentation License.

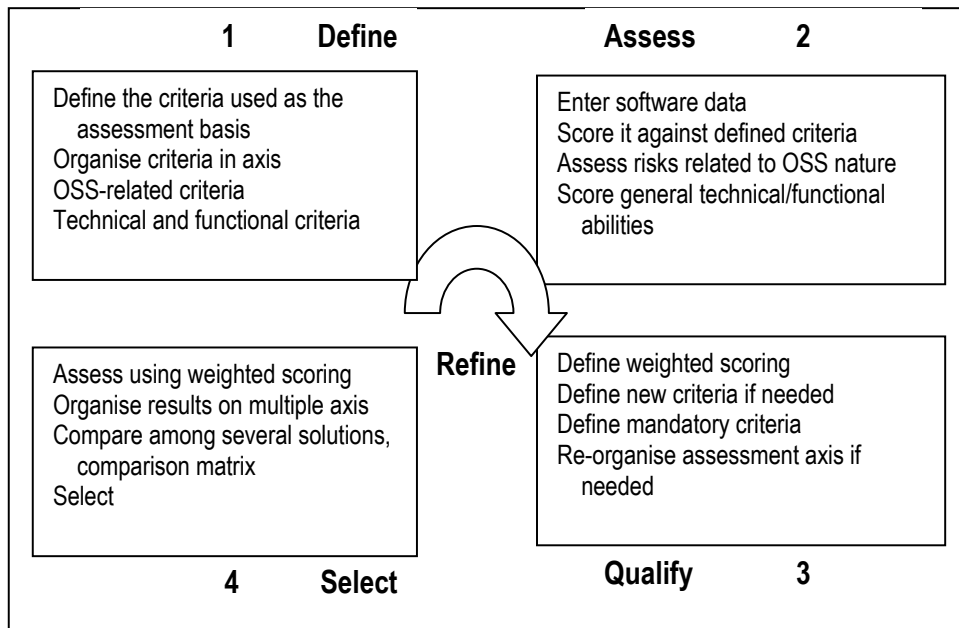


Figure 4. Four-step process of QSOS evaluation.

The method consists of four steps: definition, evaluation, qualification, and selection that are described in the following Table 5.

Table 5. Steps in QSOS evaluation.

Step	Description
1. Definition	Constitution and enrichment of frames of reference used in the following steps.
2. Evaluation	Evaluation of software made on three axis of criteria: functional coverage, risks for the user and risks for the service provider (independently of any particular user/customer context).
3. Qualification	Weighting of the criteria split up on the three axes, modeling the context (user requirements and/or strategy set by the service provider).
4. Selection	Application of the filter set up in Step 3 - "Qualification" of data provided by the first two steps, in order to proceed queries, comparisons and selections of products.

The first step, *definition*, includes defining the following sub-steps:

- Software families, what functionalities needs to be included
- Types of licenses, based on the three criteria: ownership, virality, and inheritance
- Types of communities, five types identified to date: 1) Insulated developer, 2) Group of developers, 3) Organization of developers, 4) Legal entity, and 5) Commercial entity.

The second step, *evaluation*, comprises use of the identity card and the evaluation sheet. The identity card (ID card) consists descriptions of

- General information (e.g. name, authors, references, licenses)

- Existing services (e.g. documentation, numbers of contractual and training offers)
- Functional and technical aspects (e.g. technologies of implementation, roadmap)
- Overall synthesis (e.g. general trends and comments).

The evaluation sheet includes more detailed information than the ID card as it focuses on identifying, describing and analyzing in detail each evolution brought by the new release. The main phases are 1) scoring each criterion from zero to two, 2) functional coverage being determined by the software's family and proceeding the sub-steps in *Definition*, 3) estimating the risk from the user's perspective (e.g. intrinsic durability, industrialized solution, and integration). For more information, see the White Paper at <http://www.qsos.org/download/qsos-1.6-en.pdf>

The third step, *qualification*, defines filters translating the needs and constraints related to the selection of FOSS in a specific context. Filters can be set on ID card, functional grid (concerning required level of functionality), and perceived risks from the users and service providers perspective. Filters will be defined in the O3S tool. Open Source Selection Software (O3S) is a single tool to apply the QSOS method in a coherent way. This tool is available to the community on the site <http://www.qsos.org> to coordinate creation, modification and use of QSOS evaluations.

The fourth and final step is *selection* which can be done by using strict or loose method. Strict selection is based on direct elimination as soon as software does not fulfill the requirements formulated in qualification step. Loose selection allows us to weight features and compare weighted scores against each other. The O3S tool enables the consultation of data related to a specific software and the comparison of software in the same family. This comparison is made by using weighted score patterns in a radar chart (For an example, see Figure 5).

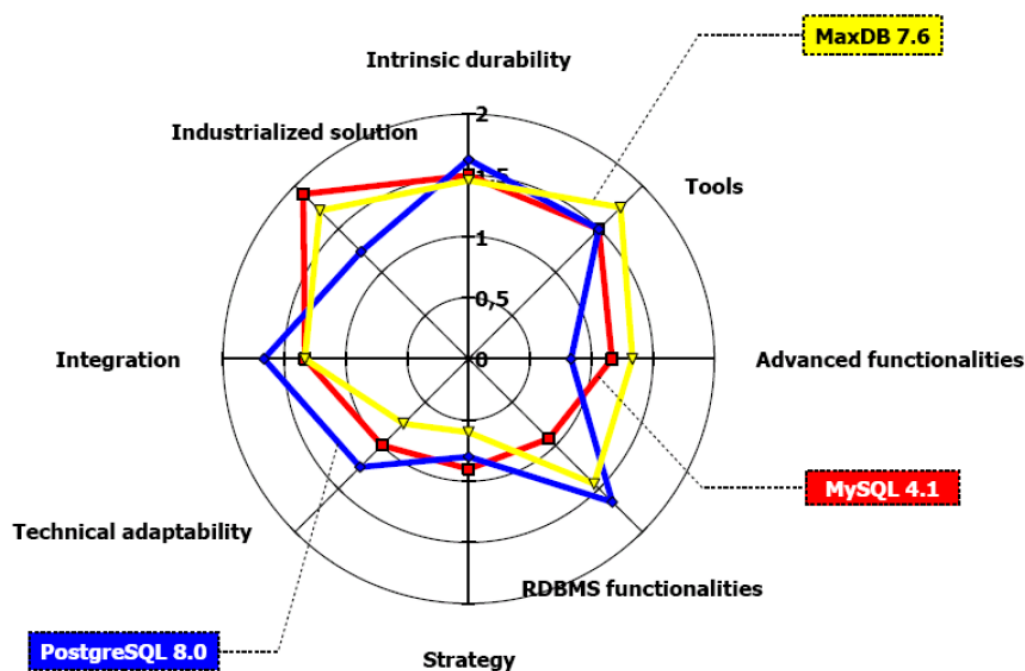


Figure 5. Radar charts illustrates differences between candidates.

Table 6. Summary of QSOS evaluation for MySQL 5.0

SECTION		Subscore	Overall
Generic			68 of 90
	<i>Intrinsic durability</i>	25/28	
	<i>Industrialized solution</i>	12/14	
	<i>Packaging</i>	19/24	
	<i>Exploitability</i>	3/4	
	<i>Technical adaptability</i>	3/6	
	<i>Strategy</i>	6/14	
RDBMS features			27 of 42
	<i>SQL compliance</i>	1/6	
	<i>Classic SQL features</i>	7/16	
	<i>Security</i>	2/2	
	<i>Transactions</i>	3/4	
	<i>Other SQL features</i>	14/14	
Advanced features			3 of 10
Tools			6 of 8
Overall MySQL rating			104 (of 150)

3.2.4 Business Readiness Rating (BRR)

Business Readiness Rating™ (BRR) was proposed in 2005 as a new standard model for rating open source software. It is intended to enable the entire community (enterprise adopters and developers) to rate software in an open and standardized way. BRR is a community initiative that is being sponsored by Carnegie Mellon West Center for Open Source Investigation, O'Reilly CodeZoo, SpikeSource and Intel. The ultimate goal of BRR is to give companies a trusted, unbiased source for determining whether the open source software they are considering is mature enough to adopt. It helps adopters to assess which open source software is best suited to their needs and enables them to share findings with the community. It promotes use and adoption of open source software and may assist developers in creating and delivering software geared to enterprise use.

The calculation employed in the Business Readiness Rating model weights the factors that have proven to be most important for successful deployment of open source software in specific settings. Among these are functionality, quality, performance, support, community size, security, and others. The Business Readiness Rating model is open and flexible, yet standardized. This allows for broad implementation of a systematic and transparent assessment of both open source software and proprietary software.

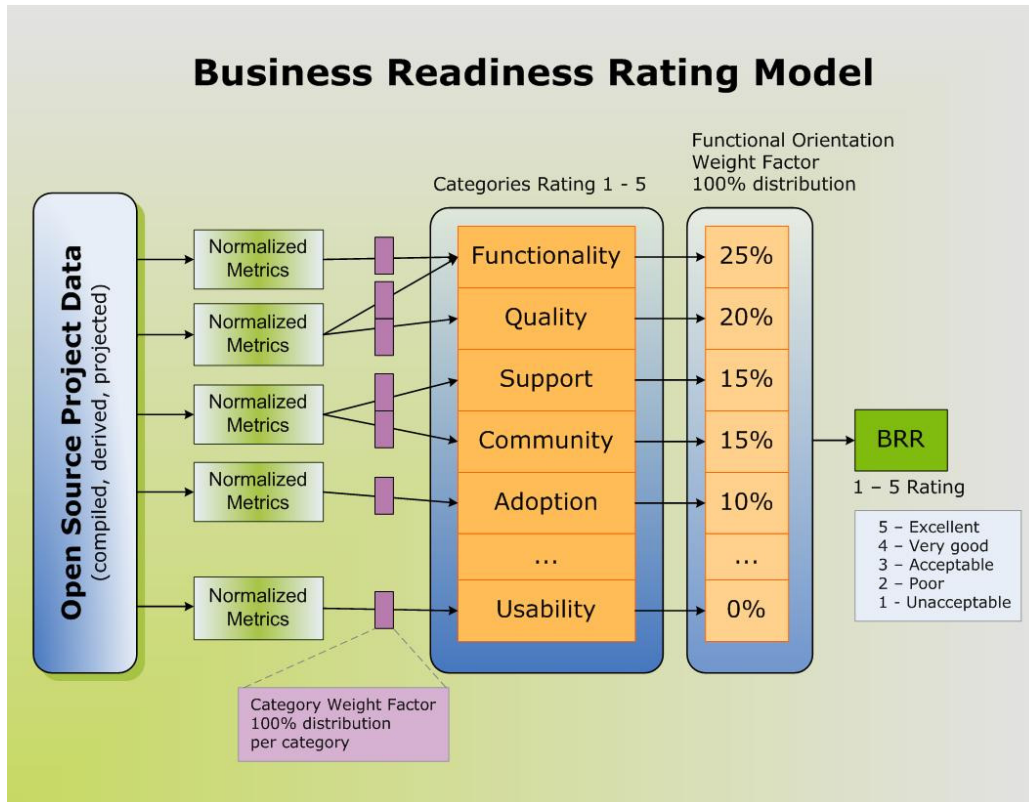


Figure 6. Overall depiction of Business Readiness Rating Model (www.openbrr.org).

The model offers proposals for standardizing different types of evaluation data and grouping them into categories. To allow adoption of this assessment model for any usage requirements the software may have to meet, the process of assessment is separated into four phases as depicted in Figure 7.

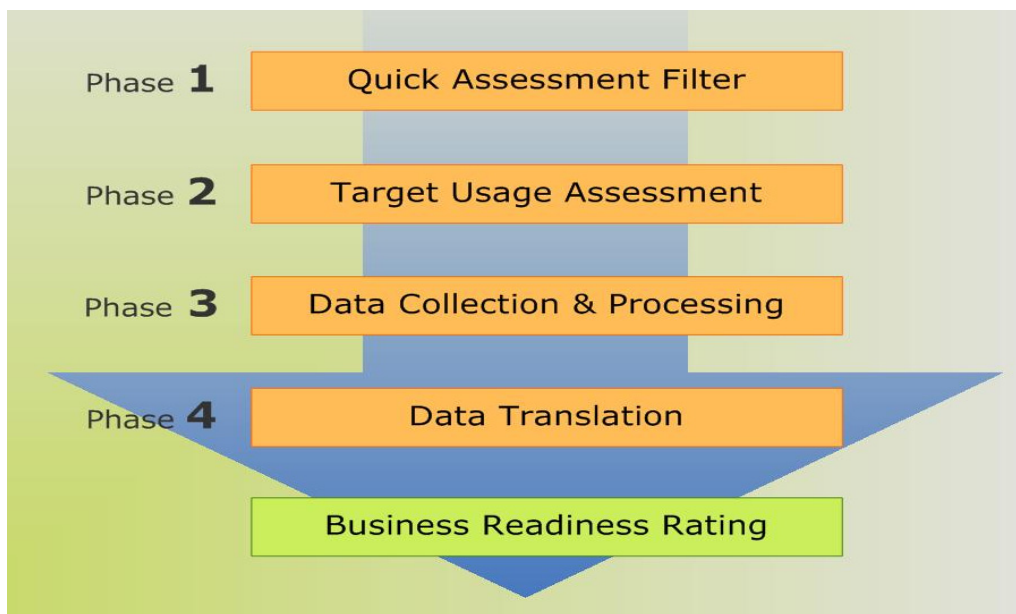


Figure 7. The Four phases of software assessment.

First, a *Quick Assessment* rules in or out software packages and creates a shortlist of viable candidates. Second, it ranks the importance of categories or metrics, third, processes the data, and last, translates the data into the Business Readiness Rating. A software component's Business Readiness Rating is scored from 1-5, with one being "Unacceptable," and 5 being "Excellent." In the following sections, the Business Readiness Rating concept and a high-level overview of how to use the model will be presented.

Initial Filtering

To assess the business readiness of an open source software component, users may start by looking at several quantitative and qualitative properties of that component. During the initial Quick Assessment phase, a simple filter lets potential adopters quickly rule in or rule out software components with confidence. Several viability indicators to use as filters in this phase include:

- What is the licensing/legal situation of the software?
- Does it comply with standards?
- Are there referenceable adopters or users for it?
- Is a supporting or stable organization associated with the development efforts?
- What is its implementation language?
- Does it support internationalization and localization in your desired language?
- Are there third-party reviews of the software?
- Have books been published about the software?
- Is it being followed by industry analysts, such as Gartner or IDC?

The list of filtering criteria for Quick Assessment is by no means exhaustive. Users may and should add filters that are important for the particular software package or situation they are evaluating.

Metrics and Categories

After completing the *Quick Assessment* process, it is important to look at which metrics and categories to use for the in-depth assessment phases. Measurable properties of an open source software project are defined as metrics. To create a standardized Business Readiness Rating, the raw data of these metrics must be normalized. Quantitative metrics, such as the number of downloads of a software package, are relatively easy to normalize whereas normalization of qualitative metrics is more subjective.

Table 7. Twelve categories for assessing software.

Assessment category	Questions describing the category
Functionality	How well will the software meet the average user's requirements?
Usability	How good is the UI? How easy to use is the software for end-users?
Quality	How easy is the software to install, configure, deploy, and maintain? Of what quality are the design, the code, and the tests? How complete and error-free are they?
Security	How well does the software handle security issues? How secure is it?
Performance	How well does the software perform?
Scalability	How well does the software scale to a large environment?
Architecture	How well is the software architected? How modular, portable, flexible, extensible, open, and easy to integrate is it?
Support	How well is the software component supported?
Documentation	Of what quality is any documentation for the software?
Adoption	How well is the component adopted by community, market, and industry?
Community	How active and lively is the community for the software?
Professionalism	What is the level of the professionalism of the development process and of the project organization as a whole?

A category rating is obtained by grouping together several metrics that measure the same aspects. How the rating in one category is calculated may differ from how another category is measured, but the results should use the same scale (1 to 5). One metric may contribute to several categories in different ways: for example, a release cycle of six months indicates a high level of community liveliness but a low level of stability.

Using the Model

The Quick Assessment phase and defining and ranking of metrics and categories according to their importance for the software's functional orientation leads us to the actions and steps taken in each assessment phase of the model to calculate the software's Business Readiness Rating.

Phase 1 – Quick Assessment

- Identify a list of components to be evaluated.
- Measure each component against the quick assessment criteria.
- Remove any components that do not satisfy user requirements from the list.

Phase 2 – Target usage assessment

Category weights

- Rank the 12 categories according to importance (1 – highest, 12 – lowest).
- Take the top 7 (or fewer) categories for that component, and assign a percentage of importance for each, totaling 100% over the chosen categories.

Metric weights

- For each metric within a category, rank the metric according to importance to business readiness.
- For each metric within a category, assign a percentage of importance, totaling 100% over all the metrics within one category.

Phase 3 – Data collection and processing

- Gather data for each metric used in each category rating, and calculate the applied weighting for each metric.

Phase 4 – Data translation

- Use category ratings and the functional orientation weighting factors to calculate the Business Readiness Rating score.
- Publish the software's Business Readiness Rating score.

3.2.5 Comparison and conclusion of the selected models

When comparing the presented four models, we may notice that all models use methods like scaling, weighting, and normalization of metrics. It might be safe to summarize that no one can get away from these techniques. However, there exist differences in how these techniques are used or defined. Respectively, all four models define assessment areas. OSMM is strict in defining its assessment areas and enforce the utilization of all areas. BRR defines 12 assessment areas, and suggest the utilization of only seven of them. QSOS uses eight and they can also be tailored for each case.

In addition, QSOS and BRR differ in how exactly the metrics are defined. QSOS is not as precise as it only mentions a few metrics. BRR tries to be specific in defining metrics and the appropriate scales for them. It has been widely recognized that if a model is too loose, the assessment power of the model will be reduced. For example, a low quality product may obtain good rating if the model allows itself to be tuned to favor the product. However, this problem is immanent in all kinds of evaluations.

BRR White Paper (2007) states that such an evaluating model should include the crucial requirements of a good software rating model as being complete, simple, adaptable, and consistent (CSAC):

- *Complete.* The primary requirement for any product rating model is the ability of the model to highlight every prominent characteristic of the product, whether favorable or not. This is necessary to prevent that the rating for any product is never misleading.
- *Simple.* To gain wide acceptance, the model must be easy to understood and relatively easy to use. Furthermore, the rating and terminology should be customer friendly. However, the model's completeness takes a higher priority.
- *Adaptable.* Due to rapid changes in the software industry, any software rating model created today may be irrelevant in the future. During the conception stage, it is impossible to capture all future potential uses of the model. Therefore, we strive to build our model with adaptability in mind — and to keep it open. That way, when the model requires an extension, it will be easy to add one without much disruption of the current model.
- *Consistent.* The scales and ratings that the model produces should be consistent across the model's different target uses. Comparable ratings for two software packages from two categories should signify equal business readiness.

We used these criteria in assessing the evaluation models presented previously. As can be seen in the following Table 8, the BRR model scores the highest overall result without any adjustments in weightings. Although the comparison can be regarded a bit superficial and illustrative, the BRR model gains the maximum points in the adaptability. As a summary, we will continue further in developing the BRR model.

Table 8. Evaluation models assessed with CSAC criteria.

Criterion	Weight	Optaros	OSMM	QSOS	BRR
		<i>Score</i>	<i>Score</i>	<i>Score</i>	<i>Score</i>
Completeness	25%	2	3	4	4
Simplicity	25%	5	4	2	3
Adaptability	25%	4	3	3	5
Consistency	25%	3	3	3	4
Result		3.5	3.3	3.0	4.0

3.3 Empirical test: evaluating Gnome with BRR model

We tested Business Readiness Rating at Nokia Multimedia. Our partner from Nokia was Quim Gil, who is responsible for Gnome-related matters at Nokia. Besides, being related to Nokia Quim Gil is a member Gnome Board, whose members are elected yearly. The board makes high-level decision in Gnome.

3.3.1 Choosing an example software

Applying a maturity model in an industrial setting requires participation from industrial partner(s). The optimal setting for applying BRR would have been a case where one of our industrial partners would have been in a position for selecting an open source component. Since such cases are not too frequent, we were forced to select the next best option: applying BRR in an imagined setting.

Our frame story was that Nokia hadn't yet selected the desktop environment to its Internet Tablet, and now they should choose among the alternatives. In other words we try to make a time trip back to the days when Nokia Multimedia was selecting desktop system to their internet tablet. Our attempt was to apply the maturity model to Gnome. Gnome being one of OSSIs four example open source communities is another argument for selecting it for scrutiny.

3.3.2 Applying Business Readiness Rating

As described in the previous Section, applying Business Readiness Rating consists of four phases. This subsection describes the phases in our case study.

Quick Assessment Phase

BRR begins with the quick filtering phase, which allows the tester to quickly abandon the obviously unviable software components. Due to the fact that, Gnome is a well-known piece of software published under LGPL, it passes this phase easily.

Target Using Assessment Phase

The next task is set the weights to the seven most important categories. The category weightings are summarized in the following Table 9 and the justifications for these weight percents are opened up in the text below.

Table 9. Summary of category weightings.

Category	Weight (%)
Architecture	20
Documentation	20
Adoptation	20
Quality	10
Community	10
Professionalism	10
Security	5
Support	5
Functionality	0
Usability	0
Scalability	0
Performance	0

Originally functionality was set as high as 15 %. However, our industrial partner was not willing to enumerate the functional requirements to the software. Therefore, we were forced to redistribute the weights so that functionality received 0%. The decision might seem rather strange. However, Quim's opinion was that functionality is important, but if some piece is missing Nokia is able to code itself the needed one. On the other hand, discovering important functionality for a desktop was hard to the researchers, too. It is quite easy to list trivial features - like the system must support various input devices - but they are included in all desktops.

Architecture, documentation and adoptation were the most weighted categories with 20% for each. From a large corporation's point of view they form such a basis for a software component, which can be tailored with respect to internal needs of the corporation. Architecture has a large impact to the rest of the categories. Having a decent architecture makes software evolution easier. The need for documentation is an obvious prerequisite for grasping the internals of the component. Having a large adoptation including other large companies ensures the continuity of the open source community. Besides, Nokia does not want to be the first-adopter.

Quality, community and professionalism received each 10% weight. According to Quim, the quality of the component core must be excellent - the rest can be fixed. The community produces the software Nokia doesn't want to develop. I.e. thriving community guarantees the future of the component. Besides, Nokia needs to interact

with someone. Professional ecosystem is a strong criterion for choosing OSS in Nokia. The other companies make contributions which are usable for all parties.

Security and support are both given 5% of weight. We are dealing with a desktop environment, in which the security is not such a big issue. However, it is not meaningless. Justification for support the Nokia should not be the one who needs support, but the one who gives it to the end users.

The usability category was given 0%, since such those problems can be fixed by Nokia. Another category receiving 0% was scalability, since it is strongly related to architecture, and thus, measuring it alone makes no sense.

Data Collection and Processing Phase

The actual data collection was carried out by the researchers with no assistance from industrial partner. In this subsection we highlight measures from the most weighted categories.

Measures for Architecture

There are three measures in this category. Gnome's unweighted ranking for this category is 5.

- "Are there any third party plugins?": Gnome goes easily beyond the limit for maximum score, which is as low as more than ten plugins. For example, site www.gnomefiles.org includes alone plenty of software for Gnome.
- "Public API / External Service": According to [OpenBRR.org] the purpose is to measure whether the product "allows for extensions via a public API, also shows design for customization". Gnome is given the maximum score with respect to this measure, since all APIs are well documented in Gnome.
- "Enable/disable features through configuration": Due to the fact that Gnome is configurable even at runtime, it receives the maximum score also from this measure.

Measures for Documentation

The two measure related to documentation are "Existence of various documents" and "User contribution framework". The unweighted ranking for this category is 4.

- "Existence of various documents": This has been properly taken care in Gnome (5 points).
- "User contribution framework": In [OpenBRR.org] the justification for the measure is that the "best guides often come from user inputs and samples, as feedback from people who have used the products". We ranked Gnome to the middle ("People are allowed to contribute" ~ 3 points) in this measure. Gnome has a wiki and user forums at gnomesupport.org. If they were filtered by "experts" then the rank would have been the maximum.

Measures for Adoption

The two measures for adoption are "The number of books at Amazon" and "Reference deployment". The unweighted score for this category is 5.

- "The number of books at Amazon": This a clever and easy-to-calculate measure, as it is carried out by making a power search query at Amazon.com with query string "subject:computer and title:Gnome". In this case the number of books is 15, which means score 5.
- "Reference deployment": This measures that through a real-world deployment, that the software is scalable and tested in real use [OpenBRR.org] Naturally, Gnome is numerously adopted, but the number of users is not made public. Therefore, Gnome is given 3 points for this measure.

Data Translation Phase

The final task is simply to compute the Business Readiness for Gnome, and publishing the rating at BRR's www-page. Gnome's Business Readiness is 4.3.

3.4 Lessons learnt

The Business Readiness Rating is an interesting opening towards a systematic evaluation of open source. The version of the method we used is not mature yet, after our trial it has been under further development. However, no new version has been published yet.

Finding the information required for evaluating the measures took roughly two to three working days. Moreover, we spent a half working day with our industrial partner. Using the BRR does not require special skills. The evaluation can be carried out by an engineer who is familiar with the application area of the software under evaluation.

One can observe similarities between software testing and BRR-like evaluation of OSS. In both the attention of the engineer is paid to small pieces of the system at a time, and then this small fragment is evaluated or tested. Similarly to the inability of testing to show that the software is error free, BRR cannot ensure the maturity of the software, but it can give us confidence when choosing open source – like when testing engineers find no hard flaws in a system makes the system trustworthy.

Unfortunately, we had no time to carry out the same evaluation we carried out for Gnome for some other desktop system. It might have given us a more elementary understanding of measuring OSS with BRR.

One problem we experienced during our trial was how to limit the system under evaluation. In other words, which plugin projects must be considered as being part of Gnome, and which are third party ones. This selection has a large impact to some measurements. For example, selecting a larger fraction leads larger amount of

Gnome-related discussion groups. This, in turn, leads to more alive discussion and better measures.

We were not comfortable for evaluating all the measures. Some measures required information that was so fuzzy, that their evaluations are simply based on educated guesses. Moreover, the developers of some measures have had quite precise model of utilized software engineering processes and tools. When this model is not directly applicable to the software under evaluation, the evaluator simply has to stretch the measure to fit in her case. On the other hand, a big amount of measures were simple and easy to apply.

All in all, using a method like BRR is recommendable when a maturity of OSS is under evaluation. The value is not only in the final result the method produces, but the process itself. It gives a structured way for investigating the software product. The final number indicating the maturity does not reveal some risks of the software. On the other hand a small number definitely reveals that the product is not mature.

The lessons learned by using the existing evaluation methods have led us to believe that a two-step evaluation is necessary (see Figure 8, below). The user-role and intended use have a tremendous effect on how the software, the community and the interaction should be approached. Not only are communities different from each other, also the user-roles necessitate various types of analysis and involve different types of risks.

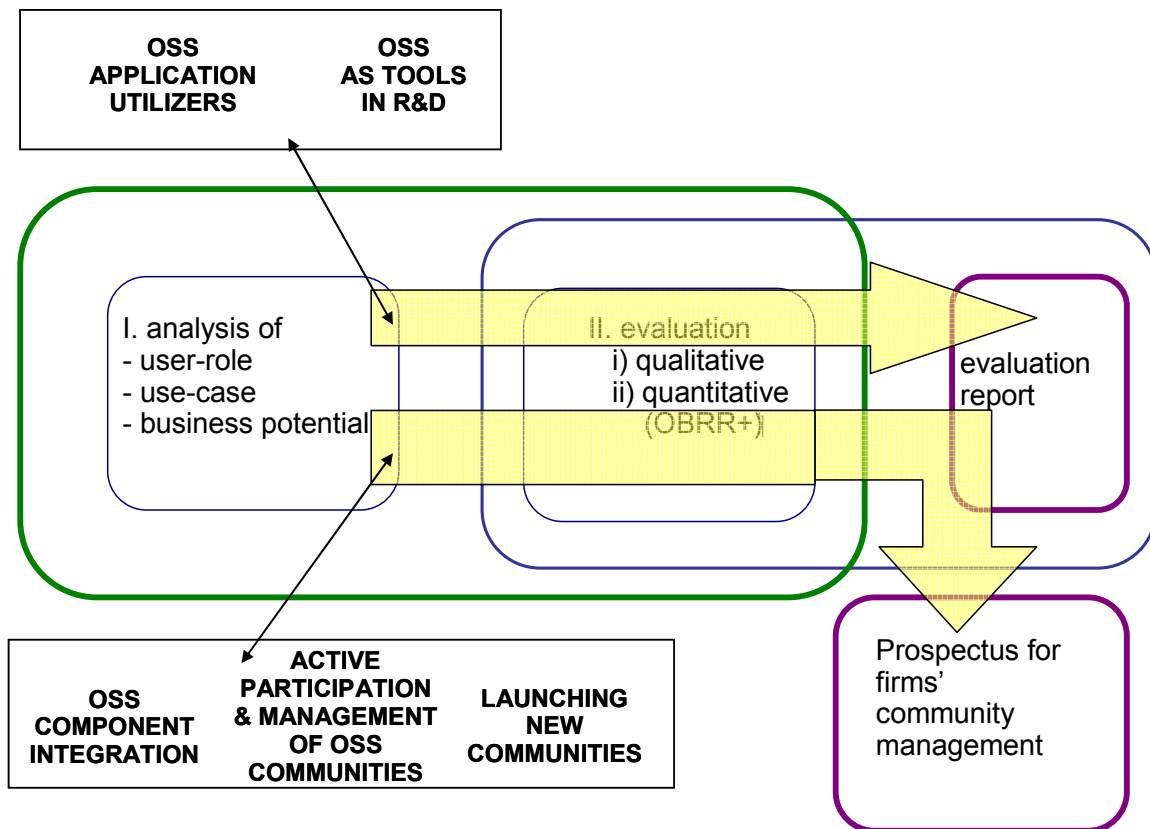


Figure 8. Evaluation process for different user roles.

Consequently, the first step of the evaluation should consist of identifying both the software or softwares to be used and, crucially, the user role of the company in question. Identifying and analyzing the user role is essential for asking the right kind of questions and identifying the possible bottle-necks and risks in the longer run. (For instance, the user-role dictates which sets of questions in the OBRR evaluation are relevant and how these sets should be weighted). By doing this it is also possible to assess the business possibilities of the use-case. So the questions asked in the first step are: i) what software or softwares exists for the task in question and what communities are behind these softwares, ii) how will the software be used and what is the user role of the company (e.g., one out of the 5 presented above) and, finally, iii) what kind of added value is sought by the OSS use (time-to-market, outsourcing, cost savings, etc.)

The second step consists of both qualitative and quantitative evaluation of the software. The qualitative analysis is performed by answering a set of questions assessing the risks (legal, economical, cultural, social) connected to the communities in question. In addition, the license checker can be used to analyze the legal situation with regard to the code. Furthermore, the communities will be classified into the ideal types discussed above, so that suitable do's and do not's can be identified as the basis of the management framework. The quantitative analysis may be performed by an augmented version of the OBRR. We feel that especially in the case of high involvement with a community the questions provided in the OBRR need enrichment with regard to both socio-cultural and technological sustainability issues.

For most cases of the two low-intensity user roles, application utilization and OSS as tools in R&D, we feel that the evaluation report produced by the qualitative and quantitative analyses will be enough. However, in the more intensive roles, from component integration to launching new communities, the evaluation report will be complemented by a prospectus for community management, consisting of practical do's and do not's, guides on best practices and long-term plans that help in organizing fruitful co-operation.

4 PART IV: CONCLUSIONS

4.1 Management guidelines for different OSS user types

In this section, we will summarize the main results by utilizing the initial framework of different OSS user types and the aspects of business, sociology, law and technology.

Firstly, the business aspect is in foci (see Figure 9 below).

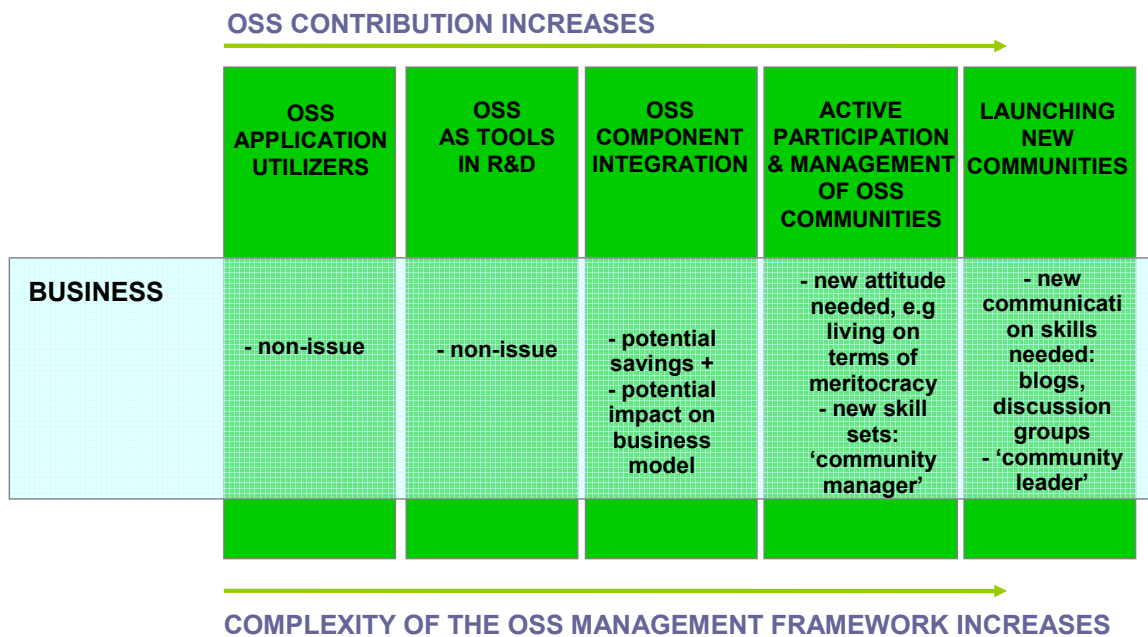


Figure 9. Business evaluation needs.

Looking at the five levels of use of open source from the business perspective, the first two ones are really non-issues from the business aspect, as it is no different from use of proprietary software.

By the third level where you are looking at the possibilities of OSS component integration, things start to get interesting. On the one hand you have the possibility of savings, but on the other hand you are approaching more critical elements of running a business where you might want to have someone to rely on e.g. product support and upgrades.

By the fourth level things start to move to a new area from traditionally run businesses, you might have to learn live on the terms of meritocracy vs. contracts, new skill sets might be needed such as 'community managers' who handle the relations with non-contractual partners.

Fifth level is the most challenging one, and this is where most of the failures take place (just consider how many of the 140,000 or so projects in Sourceforge actually

have managed to create active communities). So in launching a new community a business needs to find first of all a compelling offer where the developer communities want to partake in and secondly, but just as importantly, ‘community leaders’ from within the business that help them to make sure the development stays on track with what the company is looking for.

From the sociology aspect, the management questions will also get more problematic when moving towards the right side of the continuum (see Figure 10).

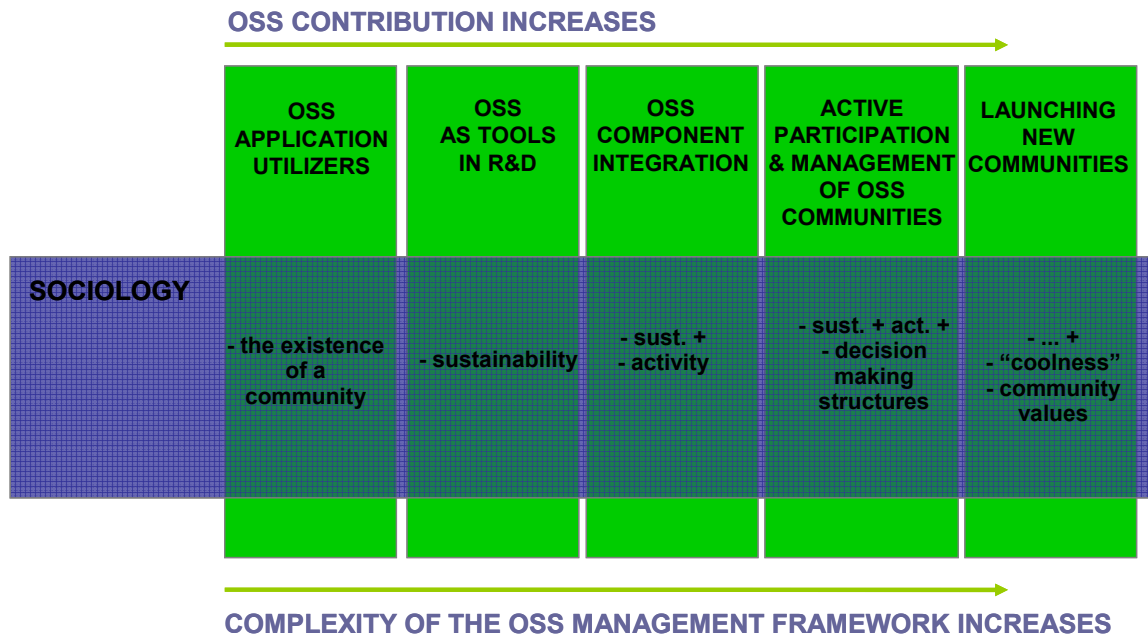


Figure 10. Community evaluation needs: sociology.

If one intends to use OS applications simply as they are, the main thing is to establish that there is a community supporting and developing the software. Otherwise one is left to one's own devices in the case that a bug emerges or that documentation or support is needed. Using OSS as tools in R&D already implicates an interest in the sustainability of the software. Here the questions presented in section 2.2. may help evaluate the longevity of the community. When we move further, the existence and sustainability of the community are not enough. An OSS component integrator will want to see an active community, one that is making progress.

When a company wants to actively take part in OSS communities, it is important to be aware of the decision-making structures in the community in question. These vary from the semi-anarchic meritocracy of the decision making in the Linux kernel community to the more explicit and even institutionalised structures of communities like Debian (with its Constitution and Social Contract) and Eclipse (with a company-run foundation). Involvement in these structures may take considerable time and effort. Finally, when launching new communities the challenge is in understanding the Zeitgeist of hacker culture in order to woo talented developers. Creating sustainable community values takes leadership, consistency and a good grip on the ideological and historical values of the developers.

From the legal aspect, the typology to different user types is not as clear, as it's always risky to generalize legal questions. However, the risk profiles are indeed quite different in different OSS-usage roles (see Figure 11).

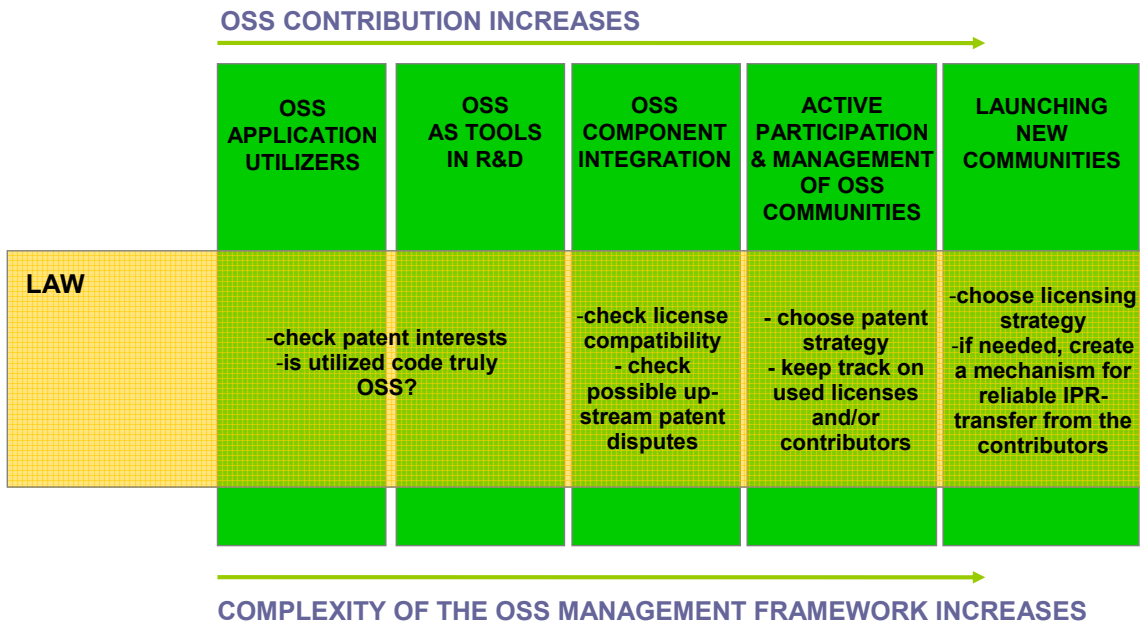


Figure 11. Legal evaluation needs.

As long as the software is used for purely internal purposes (levels one and two), the main worry is the accidental use of proprietary software without a proper license. In certain niche areas (for example vision recognition) software patents may be also something that has to be considered. However, in both of these cases the difference from using proprietary solutions from (a small) vendor is almost non-existent.

The situation changes dramatically as soon as the company distributes itself the open source products i.e. starting from level three. At that point the different rules of open source licenses take full force and it is essential to understand how different components can be mixed legally. The consequences of the mistakes are also much more direct as the distribution may now easily turn out to be a copyright violation of commercial scale. This is even more imperative if the goal is to mix commercial and open source components.

The risk profile does not change that much in level four and five. It may even decrease since now the company can have much more detailed knowledge what goes on inside a project. Establishing good legal practices (e.g. requirements who can submit code) is much easier if one works from inside. However, if something goes wrong with the project, the liabilities could be even worse than in single company situation. Another thing is that politics and legal questions are often mixed in a heated way – for example the question about the best open source license is not only a legal choice but also very much ideological one.

Lastly, the managerial questions relevant to different user types are looked from the technology aspect (see Figure 12).

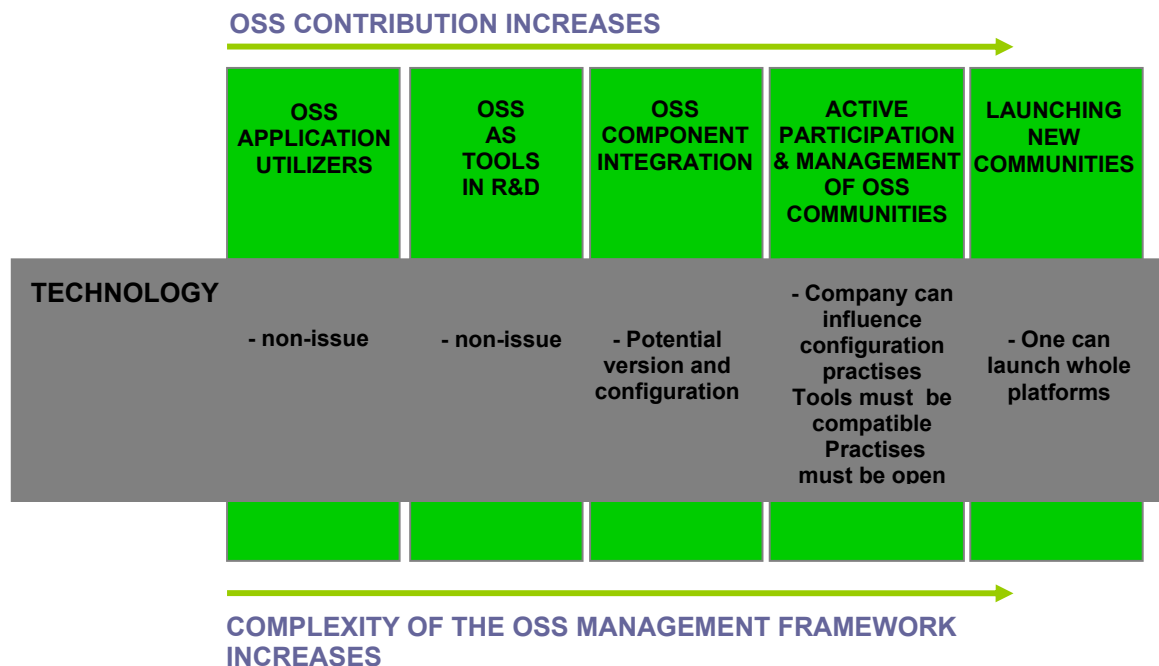


Figure 12. Technological evaluation needs.

From technical point of view utilizing open source applications and using OSS as tools in R&D does not differ from using their closed counter parts. The problems user might experience are similar and the quality of the products does not correlate with the openness issue.

The third level OSS Component Integration is the first one which matters. Having a piece of software consisting of components which are not managed by the company, sets challenges to software engineering processes and tools. For example, if the development of open source components is rapid or the changes are large, then the company is forced to use much effort for following the component development. Namely, if the gap between the current company version and the open source version is not minded, then the company will be in trouble in future when the new version of the component is integrated.

If the company is in the fourth level Active Participation & Management of OSS Communities then the company is able to impact the development practices. This might alleviate the potential problems described in the previous paragraph. The company must use the tools which have been chosen by the open source community. Moreover, the practices should be open, so that they can be easily integrate the practices utilized by the community.

Being in last level Launching New Communities indicates the maximum contribution level in our taxonomy. In the technical sense, the company can now launch whole platforms including the core system and SDKs for developing third party software for the system. After the launching the company is quite dependent of the community, and it has (at least moral) responsibility for the community.

4.2 Evaluation of the results and insights for further OSS research

In this report, the main results of the OSSI research project have been presented, forming the OSS management framework. The framework rests on three closely interrelated issues:

- the importance to understand different community types in order to interact successfully with/within them
- the need for appropriate evaluation tools and processes in order to select the best alternative for software production
- the technological, legal, social and business risk management guidelines for different types of OSS utilization

Together these issues give us the holistic view, which is needed to successfully operate in OSS field.

The developed typology of different OSS communities and the best practices to operate with these different community types are rather detailed and thus provide a step forward for firms to gain a sense of manageability, or at least ability to influence, on communities. However, the guidelines for community best practices are based on our research of the four case communities, Eclipse, GNOME, Debian and MySQL, thus the generalizability of these guidelines is naturally limited.

The review of the existing OSS evaluation tools and the comparison between them, contributes to the OSS literature, but hopefully also gives tips for firms to select the most suitable evaluation tool for different situations and contexts. Moreover, testing of one selected evaluation model, the BRR, and the lessons learnt from it, may give useful guidelines for firms when they carry out an evaluation process. Based on the comparison of the models and the test of the BRR model, we developed a new evaluation tool, TSOSSA, presented in chapter 3. This TSOSSA tool still needs further development and verification through extensive empirical studies, which we will continue in our further research projects.

The third keystone of our results, the framework of the different OSS utilization types, served us nicely during the research. The framework helped us to better grasp the essence of OSS business, and enabled us to provide more detailed OSS risk management guidelines that would have been impossible without this kind of OSS utilization typology. However, during the research we also found out, that the framework has its own shortcomings, too. Especially when it comes to the issues of increasing OSS management complexity. For example, one might argue that producing your own communities is actually less work than the previous phase in the framework in some cases, but can at first appear to be the opposite. Thus, for the analytical purposes of the OSSI research project the framework was functional, but for further research and practical management one should keep in mind the dangers of making such clear-cut categories of OSS utilization types.

For future OSS studies, we argue, that the increasing use and development of OSS is rapidly multiplying situations in which a) companies involved in OSS use and

development need to evaluate an existing piece of OSS software and the project behind the software (whether it is a volunteer community or a company driven project) and b) a public sector utilizer (such as a school, municipality, state agency, university) needs to make an informed analysis of and long-term commitment to an OSS application or platform. Both of these cases necessitate the existence of readily available and scientifically grounded evaluation methods and tools.

In particular, evaluation from the point of view of a company needs to take into account the spectrum of different user roles and the level of involvement with the community that these imply. Also, many companies are already familiar with the use of OSS, and the need for a standardized evaluation method is prompted by a continuous need to stay abreast of development. From the point of view of the public sector, a crucial challenge is the vertical integration of various sectors working together but having a widely varied history of ICT use in terms of legacy systems, commissioned software and tailored integrations. Consequently, the OSS evaluation has to concentrate on issues of compatibility, sustainability and customization, and the software providers can be expected to form alliances that serve these needs.

Furthermore, we argue that studies, which will take a closer look on the OSS utilization from the different decision-making levels within organizations, would make a great contribution to the needs of business practitioners. We have already stated in this report, that the management challenges are different in different types of OSS utilization, but we argue further, that the problems to be solved and issues to be focused on, vary whether we look it from the level of a company, division, SBU, single project, team etc.

Although many of these avenues for future research would contribute especially to managers and companies facing the practical challenges related to OSS, a potential for drawing up from these context related questions on to more general level of management literature clearly exists.

REFERENCES

Golden, B. (2004). *Succeeding with Open Source*: Addison-Wesley.

Himanan, P. (2001) *The Hacker Ethic*. New York: Random House. Karim, R Lakhani & Robert, G. Wolf, 2005. "Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects," at Feller, J et al. (edit). *Perspectives on Free and Open Source Software*. Cambridge, Massachusettes, London: The MIT Press.

Lash, S. (2002). *Critique of Information*. London: SAGE. Lawrence Lessig, 2004. *Free Culture*. London: Penguin.

Levy, S. (1984). *Hackers. Heroes of the Computer Revolution*. London: Penguin.

Mikkonen, Teemu., Vadén, Tere. & Vainio, Niklas. 2007. The Protestant ethic strikes back: Open source developers and the ethic of capitalism. *First Monday*, volume 12, number 2 (February 2007)

OpenBRR.org. (2005). Business readiness rating.

Optaros Inc.

http://www.optaros.com/en/publications/white_papers_reports/open_source_catalogue_2007

QSOS <http://www.qsos.org/>

Raymond, E. (1999). *The Bazaar and the Cathedral*. Sebastopol: O'Reilly.

Reidar Conradi, Amarjit Singh Marjara, Øivind Hantho, Torbjørn Frotveit, and Børge Skåtevik, "A Study of Inspections and Testing at Ericsson, NO", a rewritten edition of the paper presented at PROFES'99. Oulu, Finland, June 1999.

Woods, D., & Guliani, G. (2005). *Open Source for the Enterprise. Managing Risks, Reaping Rewards*. O'Reilly.

APPENDIX 1: Open Source Business Reflections

Mikko Puhakka

Mikko.Puhakka@gmail.com

Jukka Ala-Mutka

Introduction

As open source and open source business is rapidly becoming mainstream as witnessed e.g. by the latest estimates from IDC, Gartner and seemingly most analysts it felt that it was an appropriate time to try to get some feedback from entrepreneurs and experts in this space about some of the do's and don'ts they have learned over the years building open source businesses. All the answers can be found collected in Mikko Puhakka's blog at <http://blogit.digitoday.fi/opensource/2007/06/15/> , but here are few select ones.

Reading through them one will find that while some reflections reflect very traditional business wisdom, others especially building and dealing with the open source communities and transparency in open source bring new elements to building and running a business.

Expert Comments from summer of 2007

"A superb team is a must in all start-ups. The special thing with open source startups is that they may have a history as an open source project before becoming a commercial business. The team must understand how to master this evolutionary transition."

"The strength of open source lies in massive participation by users worldwide. It takes special dedication and skill to build an architecture of participation."

"It's dangerous to think that open sourcing something will solve all problems. It probably won't. Open source accelerates what would happen anyway. If you have a crappy product, it will die sooner if you open source it. If you have a great product, it will succeed sooner if you open source it. Open source also is not a business model in itself. You have to figure out the business model as a separate exercise (there is a handful of good alternatives)."

Mårten Mickos, CEO MySQL

"Your community of users is an incredible asset to spread the word. It's not just about people using your software for free and telling other people about it, but rather the fact that developers will start taking it to work and it will sneak in under the floorboards. This is how the PC revolution started. It's

why Visual Basic is still huge. It's how the Linux revolution happened. So too with MySQL. And then the CIO discovers it and they need to treat it as a proper product asset just like any other asset on which the business depends."

"Understand your value proposition and your core competency, and choose your license wisely: if your entire core competency that enables your core value proposition to your customers is embodied in the software, DON'T publish it in such a way that you give away the company. I have seen a situation in the security world where the software solution was everything. If they had made the software available under the wrong license, they would have essentially given away their future growth. Just because you published the source code does not mean the world is going to work for you for free. It's been a while since we saw this level of naivety with the original Mozilla launch from Netscape, but I'm betting there are still a lot of business people that don't understand open source software economics that still have old ignorant opinions."

Stephen Walli, an Open Source expert

"Open source is the natural step in the evolution of software development within an Internet-centric environment. Or put it another way, it is software development 2.0. Thanks to collaboration through Internet communities it is possible to challenge large well-established companies with a small team of enthusiasts locked in a garage in the most remote corner of the world. It is possible to do so with very tiny budgets that any average geek can afford. It is also possible to accelerate it with relatively small seed investments, not only from VCs."

Ignacio Correias, CEO of Warp

"The relationship that open source projects and vendors have with their community often extends beyond - sometimes far beyond - the traditional buyer/seller interaction. Like any serious relationship, though, it requires more work than one that's shallower in nature."

We recommend asking yourself the following question on a regular basis: how can you better serve your community? What can you do to help? Not to the detriment of your own enterprise, of course, but there's always more that can be done. Maybe you can provide office space for meetups. Maybe some pro bono legal counseling. Maybe you can use your contacts and network to try and resolve issues. Maybe you can employ developers to work on a project that's aligned with your business interest. Maybe you can sponsor travel for a developer to attend a conference."

Look around, you'll find something. Don't think of it as altruism, if that makes you uncomfortable: think of it as pragmatism. It's kind of a truism that you get out of relationships only what you put into them; in my experience, open source is no exception."

“Do Not Forget Questions of Project Governance: In the world of open source, license choices often get an undue share of the credit or blame for project success or failure. Though I do not subscribe to the notion that the license is unimportant - I think indeed it matters very much - project governance and the implications of that governance are at least as important if not more so.

The choices made with respect to governance (and the tools that support it) can impact whether or not you'll be able to accept outside contributions (thus amortizing the cost of development across external parties), the volume and quality of contributions you might expect back, the relative strength and goodwill of your community, the adoption rate you can expect to see and so on. Its reach is, in other words, quite profound, and yet it can often be an afterthought following licensing conversations and choices.”

Stephen O'Grady, Analyst at Redmonk

“Focus on real value. The value of software bits is going to zero, if it's not there already. The worst thing a business can do is to put its head in the sand and pretend that the world hasn't changed. It has. Of course, like the real estate market in Florida (either in the 1920s or the present era), or the Internet Stock bubble, there is money to be made by people who are luckier than they are smart. But if you know that the value of something is going to zero, why buy in with the belief that you can find somebody more stupid than you before the value does go to zero. The freedom that comes from knowing that the bits are worth nothing promotes one to focus on what is of real value:

Is it helping customers increase revenues?

Decrease cost?

Solving the business problem one is paid to solve?

Delivering on time and on budget?

The ability to be more flexible and responsive than your competition?

Delivering a quality product?

Creating the kind of customer loyalty that enables new problems to be solved?”

“The goal of Sarbannes-Oxley is transparency, and yet many provisions and requirements of that act are so burdensome that the benefits of transparency are lost in a quagmire of reporting requirements. By contrast, open source software provides a natural transparency that can become pervasive. Open source can encourage institutional transparency by acting as a constant reminder that secrecy, more often than not, is the refuge of the incompetent. Again, many entrepreneurs and many companies live in a constant state of denial, be it about their employees, their products, their quality, or their strategy. Sufficient transparency encourages one to actually address problems rather than putting energy into hiding them.”

Michael Tiemann, President of Open Source Initiative

“Open Source users are more open to new and innovative functionality and they are downright eager to participate in the end stages of its development. There is a fine line to walk here, but essentially you can usually release much earlier betas of your product and get a wider set of usage scenarios vetting the quality of the software much faster than in the proprietary world where software hides behind a dark curtain of mystery until its released. Of course, this is not to be abused! If your betas are inherently broken the community will shun them. Ultimately if used right, the company and the community benefit from faster time to market for product development.”

“Although surprisingly common, if you base your software business model on services alone for revenue, or even greater than 25% (completely arbitrary number), you are in trouble. One, its going to take you a LONG time to get critical mass of deployments. Two, its going to constrain your growth to hiring an army of professional services who you trust to be consistently as smart as you and as versed in your project. Three, its likely going to slow or stop your ability to generate real improvements to your project. Fixes may happen in the context of specific use cases but real innovation and extensions are going to be highly subject to a slow period in services... during which you aren't making any money or growing! Not to mention you will likely be closer to airline staff than your own family, because of the rigorous travel schedules.”

Hyperic CEO Javier Soltero

“Do something revolutionary: software is more fun and more dangerous (to the other guy) when it is really revolutionary. Not merely innovative- ‘we want to do something new’ but revolutionary - ‘we want to actively overthrow the old.’ We are here because we revolutionized the method of software production, but to continue to grow, we have to start revolutionizing other things too Be the Wii not the PS3. If you really want to catch them by surprise, be a revolutionary using open source, but not in software. The people outside software (well, except for poor Encyclopedia Britannica) don't yet realize what is coming for them.”

“Don't let your community's fears drive your feature choices: if you're a real open source company, you'll have very direct contact with your customers. This is normally a great thing, but when you make a decision they don't like, particularly if it scares them, you'll hear about it loud and long and clear. If you let that negative feedback drive your decision making, you'll never grow beyond the needs of those people. You must not be afraid to piss them off when you truly believe that a design decision is for the broader good. Remember, the pissed off people scream- the happy people just go on with their lives. So you can't just say ‘more people screamed than thanked us’- that isn't a useful metric.”

Luis Villa

Conclusions

In building an open source software business you need the same basic elements as in proprietary business: a great team, good value proposition, good project management and so forth.

By open sourcing a product a bad does not turn into a good one, so there is no magic in it. General consensus seems to be that the value of the actual software or bits is approaching zero whether you are providing open or closed software. So you need to find value and revenue from services, support contracts, providing solutions etc. Open source does provide a way to challenge incumbents, but it has to be something more than just opening up the code, that does not provide enough of disruption.

To cause disruption and create successful business you need to learn to create and compel the developer communities on something that is of value to the business, some of these things might seem very unorthodox to business leaders but in order to succeed in open source business, the skills are necessary. You have to e.g. learn what are meaningful ways of contributing back to the community that is working for you, otherwise, why would they build you a business for free?

While these commentators certainly are pro-open source the fact that they are either running or working with successful open source companies certainly brings credibility to their statements.

APPENDIX 2: Introduction to License Checker

Case Study in Software License Management: Open Source License Checker Tool

Jing Jing

jing.jing@hut.fi

Sakari Kääriäinen

sakari.kaariainen@hut.fi

Abstract

Open source software, as part of the fast growing digital economy, possesses unique features from both business and legal aspects. This paper describes the legal issues of open source including open source licenses, associated legal risks and management of these licenses as well as solutions for managing the license compliance issue in open source software. In the risk management solution section, we discuss the existing commercial license compliance solutions and describe an open source solution for managing the open source license compliance issue.

1. INTRODUCTION

Since the 90s, digital economy has demonstrated a rapid growth along with the fast technology development. In the 21st century, we see that the digital world is gradually integrating into our substantial world and becoming an important part of the environment where we work, play, and live everyday.

1.1. Background

In the digital economy world, there seems to be endless amount of technology innovations and business opportunities. It becomes more and more evident nowadays that the technology innovations are recognized as core competence of companies, which the business is based on. What is tightly linked with the technology innovation is intellectual property, which has become the intangible assets for individuals and companies these days. IPR (Intellectual Property Rights) *defines the rights that allow people to own their creativity and innovation in the same way that they can own physical property* [1]. In the recent years, it has been widely adopted as a business strategy for individuals and companies to preserve the value of these creations from unauthorized use and provide the incentives for further development.

When mentioning about IPR in the software industry, the most common term we hear about is software license. Software license is basically a form of contracts, which *defines the permission to perform some act otherwise would be harmful* [2]. No matter if the software is proprietary software or open source software, licensing issue is equally important for both. So much effort has been paid on the license issue related to proprietary software in the recent years. For those not familiar with the concept of open source, people presume based on the name and their experience via using open source that the licensing issue is not as important as that of proprietary software.

However this is a misguided perception of open source. As the popularity of open source is growing fiercely, open source licensing starts attracting attention from both individual and enterprise audiences.

Licensing of proprietary software and open source software stands on substantial different grounds. *The fundamental purpose of open source licensing is to deny anybody the right to exclusively exploit a work.* [3, page 4] This is in principle contradictory to proprietary software licensing where the users are not allowed to make copies for others, derive the work or authorize others to do so. Open source not only allows redistribution of the work, it also promotes modification of the original work.

However open source licensing, which in a sense is no different than other type of licenses, also poses certain constraints. The most known limitation is that under certain licenses, the redistribution as well as the derivative work must be under the same license as the original work. One of the key risks associated with this limitation is that *large open source software packages may include components whose licenses terms are incompatible with the rest of the package* [4].

The fundamental question is – how can we manage the open source licensing and its associated risks so that we can make better use of open source software?

1.2 Objective of the Study

As open source has been nowadays widely adopted both by enterprise and personal users, the attention to the risks of open source licensing has been rising. The motivation of the study is to demonstrate our understanding of the open source licenses and their related legal risks from the license compliance perspective. The objective is to present existing license management solutions and introduce an alternative solution developed in a research project in Helsinki University of Technology.

1.3 Research Questions

Based on our motivation and objective of the study, we aim to answer the following research questions:

What are common open source license types?

What are the risks related to open source licensing?

What are the existing commercial solutions to manage risks in regards to open source licenses?

What is the open source alternative for managing license compliance issue?

What is yet needed to study and solve?

1.4 Scope of the Study

This study combines two hot topics in the present world of digital economy: *open source* and *license management*. Open source is a rather wide area to discuss, however we aim to discuss on the license management aspects of open source. In this paper, the key focus of the discussion is the risk management of license compliance issue and how tools can support this management process.

We discuss the current open source licenses and commercial tools, which automate the software license management process. More importantly, we want to introduce our research result – an open source license analysis tool “Open Source License Checker”, which could be useful in the license risk management process.

1.5 Methodology of the Study

In the beginning, we present the literature review of existing open source licenses and the associated legal risks to manage open source licenses. Then case studies on commercial license management software are demonstrated. At the end, we present our open source solution developed throughout our research project aimed to manage the open source licenses.

1.6 Structure of the Paper

Chapter 2 provides an overview of the open source licenses, the associated problems and risks, as well as the existing commercial solution for software license management.

Chapter 3 introduces Open Source License Checker – an alternative open source solution to manage the open source licenses. The feature of the application and the technical description is included in this section. The future improvements are also included.

Chapter 4 presents and summarizes the research results in this paper. Based on this knowledge, additionally we open the discussion on the future of the open source licensing and risk management issues.

2. OPEN SOURCE SOFTWARE LICENSING

2.1 Overview

The spirit of the open source software – openness and freedom has great impacts on how the open source software is developed, managed, and licensed. The most noticeable difference to proprietary software is that open source allows free redistribution of the program and modification to the source code, which is strictly forbidden by the proprietary software.

By September 2006, there are 58 open source licenses approved by Open Source Initiative [5]. In the next section, we introduce briefly the types and the key characteristics of commonly used open source licenses.

2.2 Open Source License Summary

Based on the description above about open source licenses and according to [3, page 34] there are two main classes of open source licenses: BSD-style licenses and GPL-style licenses.

MIT (or X11) license is probably the only open source license, which poses almost no restrictions to the licensees in using the work; whilst BSD imposes certain terms on the distribution of the original and derivative works. However, a distinctive feature of BSD-style licenses is that they allow the code to be “closed”; that is modified and published under a proprietary license without including the source code with the binary program. Some view this as a contradiction to the idea of open-source software, but the benefit of allowing commercial derivations is that the software might have wider influence. For example TCP/IP stack that was part of BSD Unix became the basis of Microsoft's stack implementation. It's possible to combine BSD license with almost any other source license. MIT license is very similar to the BSD license. Apache license can also be combined with proprietary software. The BSD-style licenses are suitable for situations when software is wanted to be widely used even as a part of proprietary software.

Another major class of open source licenses are GPL (GNU General Public License)-style licenses. The main difference between GPL-style and BSD-style licenses is that GPL allows derivative works to be distributed only with GPL license. No changes to the license text itself are usually allowed. This makes it difficult to combine GPL-licensed code with proprietary code (although it might be possible if open source components are not statically linked with proprietary components). GPL-style licenses are not always compatible even with other open source licenses. LGPL (GNU Lesser General Public License) and Mozilla Public License work in a similar way.

2.3 License Compliance Issues and Risks

In this section, we use GPL license as an example to explain the license compliance issue and risks.

A lot of discussion has been on-going regarding the license compliance issue with open source software. One of the well-known case is when D-Link infringed the GPL licensed Linux kernel code and other programs which are running in a network attached storage (NAS) product. [6] Many more GPL violation cases have been uncovered afterwards. This violation is rooted to the nature of the GPL license which does not allow the redistribution and modification of the original code to be under other license than GPL itself.

Based on the limitation with GPL-style license, it is obvious that there is a license compliance problem not existing only between the GPL licensed software and proprietary software, but happens also amongst open source licenses themselves. According to Rosen [7], *open source code licensed under one approved reciprocal license may not be used in a project licensed under another approved reciprocal open source license*. A typical example would be that a GPL licensed software has been modified and redistributed under BSD license. This implies that this software under BSD license can be even changed later to proprietary licensed software. This is specifically against the initiative of the GPL license.

License problems can also happen when code under proprietary license is included into code under open source license. This not only puts the open source community at risk, but could also cause significant damage to the proprietary software company who owns copyright or patent to the original code. Not mentioning the damage fee ordered by the court to pay up, the cost of hiring a lawyer could already bring very bad damage to most small to medium open source communities.

It is clear that license compliance issue introduce legal risks to both open source society and proprietary software companies. According to the risk management, there are several action levels to handle risks. The best choice is always try to avoid them. However this is not always easy and possible to achieve per say. In most of the cases, we confront and minimize the potential risks and try to manage them through time by adjusting our actions and decisions. The worst case for risk management is the contingency management. In this paper, we only focus on discussing how to avoid, confront and minimize risks, so that we could avoid the contingency management process.

2.4 Commercial software for license analysis

In this section, we present two commercial software license management tools with the functionality to manage license compliance. We also discuss strengths and disadvantages of these solutions.

2.4.1 Black Duck¹ Software Compliance Solution

Black Duck, a private held company, was founded in year 2002. It is currently a successful software company offering commercial software compliance solutions for enterprises.

Black Duck has two key products. exportIP is a software which automatically manages encryption in software-based products and comply to the US export law. The protexIP product provides license compliance solutions for software companies to manage how software are created, managed and licensed. In this paper, we only focus on studying the license compliance solution - protexIP. The encryption management solution is out of the scope of this study.

The protexIP solution is primarily based on Code Print technology which compares the developed software code against thousands of other software projects. This is a software program code comparison based license compliance management tool. KnowledgeBase is a database in which protexIP stores thousands of open source and commercial software projects and it can be constantly updated. Based on this core technology protexIP implemented a series of comprehensive license compliance features.

Black Duck protexIP has well established its position in the software license compliance market. Many companies have chosen this solution for managing its software products. However, Black Duck does not provide a trail version of the application which allows potential users to get an initial experience on the software. In addition as it is a commercial closed source software product, the matching and analysis logic has not been published for users to verify how the license matching and the code checking are implemented.

2.4.2 Palamida

Palamida² is a company launched in 2003. They currently offer two software products: IP Amplifier³ and IP Authorizer⁴ in addition to auditing services⁵.

IP Amplifier is a software product for analyzing license information in source code packages. According to Palamida's webpage and the IP Amplifier datasheet⁶ major features of the product are:

- A monthly-updated database that contains license and copyright information, code snippets, binary files, java namespaces and product descriptions of 750000 commercial and open source projects. It's not specified how much information about an average program is actually stored in the database.
- Multiple scanning methods for identifying third-party software components
 - Source code detection: code snippet matching against the database (works with any programming language)
 - Binary file detection: digest-based matching of binary files against the database

¹ Black Duck home page – <http://www.blackducksoftware.com/>

² Palamida company home page: <http://www.palamida.com/>

³ Palamida IP Amplifier description: <http://www.palamida.com/products/ipamp/overview>

⁴ Palamida IP Authorizer description: <http://www.palamida.com/products/ipauth/overview>

⁵ Palamida auditing services: <http://www.palamida.com/services/audit>

⁶ Palamida IP amplifier data sheet: <http://www.palamida.com/pdf/IPAmplifierFall2006.pdf>

- Namespace detection: Java and C# namespace matching against the database
- License detection: license text snippet matching against the database
- Copyright detection: detection of copyright information inside source files
- User specified search: detection of user-specified text inside source files
- both rich client and web client
- integrated scripting language
- exposed API's (programming interfaces)
- compatible with ANT, Make and "all major source code management systems" (it's not specified what source code management systems this actually means)

Unfortunately no demo version of the product is available so these claims could not be verified. Compared to the OSLC tool, IP Amplifier seems to contain much richer set of functionality but it's impossible to evaluate how well the actual license detection works compared to the OSLC without practical experiments. No details about the matching algorithms are given in the Palamida's webpage. Also no further information was found in the web or in the scientific literature.

System requirements of IP Amplifier are very steep: recommended specifications for the server component are 300 GB disk space and 12 GB memory. For rich client application 2 GB memory is recommended. This should be compared with OSLC program that doesn't require any server component and runs fine with machine having 512 MB memory.

3. OPEN SOURCE LICENSE CHECKER TOOL

3.1 Overview

Open Source License Checker (OSLC) tool is one of the key deliveries of a research work done in Helsinki University of Technology for a research project - managing Open Source Software as an Integrated Part of Business (OSSI) [8]. This project involves 4 Finnish universities and 10 companies, and is primarily funded by Tekes – National Technology Agency of Finland.

The initiative of developing this tool is to provide a unique and reliable solution for managing open source software license compliance which does not yet exist in the market. The Open Source License Checker analyzes license information by extracting all license information from open source packages, comparing them to the original license text from the license database, and summarizes the overall license information from the package.

Differentiated from the existing commercial license compliance software, the Open Source License Checker is an open source implementation under GPL license. Most of the license compliance software is under commercial license, which is not affordable for many open source development projects as not many of them are profit driven. The open source license checker offers an alternative, economical yet reliable contribution to the open source society to address the software license compliance issue. Moreover, the open source approach provides end users the visibility and freedom to check out and modify not only the implementation of the code, but also the license database whenever needed.

The objective of this is to provide both the management and the development a solution for managing open source licensed software. From the management aspect, the tool can be used to find the licenses existing in the package, the matching to the

original license text and the incompatible license information by presenting the result of the analysis work from the whole software package. The management could use this information for instance to choose the best suitable software packages for their software development and also to decide the type of license for the software under development. This could directly avoid the legal risks even when the software development starts. While during the software development, engineers can use the tool to check which files in the source package are problematic and take corrective action to make sure that all source code is license compatible.

The current stable version is stored in sourceforge.net [9]. It was developed and managed by researchers and students from Helsinki University of Technology.

3.2 Features

The Open Source License Checker is implemented in Java and has passed the testing on major operating systems, such as Windows, Linux and Mac. It provides both graphical user interface as well as command-line user interface.

Common features provided by both interfaces are:

- Access to zip, jar and tar packages as well as file system directories
- Identifying open source licenses from:
 - o Java, PHP, and C/C++ source files
 - o Linux kernel source files
 - o “LICENCE” files
 - o “COPYING” files
- Indicating the license matching confidence against the original license text
- Highlighting the matched license text
- Displaying source code import references
- Link to import files (only in Java)
- Displaying the license conflicts
- Identifying license exceptions & forbidden phrases
- General summary and report on the source files in the package
- File filtering in source package based on different criteria

The GUI version has the following additional features:

- Source file print support
- Browse history support
- Complete Help functionality

3.3 Business Benefits

Blackduck’s protexIP and Palamida’s IP Amplifier for managing software license compliance both demonstrate a set of comprehensive features. As license identification is the essential feature and can be recognized as the core competence of the software, the reliability of the identification is extremely critical. However as both products are proprietary software, it is hard to verify the core technology. OSLC, under GPL license, not only offers free software but also free source code for users to verify how the core technology is implemented. In addition, users are allowed to

modify and improve the program based on their own needs under the GPL license terms.

From the budget aspect, both commercial products could pose a financial challenge to small to medium size companies. On the other hand, most of these small companies need to deal with software license issues in their management and development, and do not afford time, resource, and money to lawsuits. OSLC in this context can offer an alternative solution to the commercial product for managing software license compliance issues.

Most of the time, when talking about software licenses, we think that it is a task related to lawyers. The reason is that software licenses are written in legal languages and it is indeed difficult to read and understand. Therefore it is better to be handled by a lawyer. However in a software company, this is not the case. For the business managers and software engineers, their priority is to develop and extend business, but licensing is never an issue they can avoid. The OSLC program could provide the following benefits for both the management and the development:

- To identify and present the license analysis result from a software source package
- To select open source software for development
- To avoid legal problems and lawsuits
- To manage software package with multiple OS licenses
- To support decision making for OS software license
- Work more efficiently:
 - o To save managers' and engineers' time from going into details in the source package
 - o Managers and engineers do not have to be open source license expert

3.4 User interfaces

3.4.1 Graphical user interface

The graphical user interface is presented in the figure below.

Figure 1: OSLC overall package license information

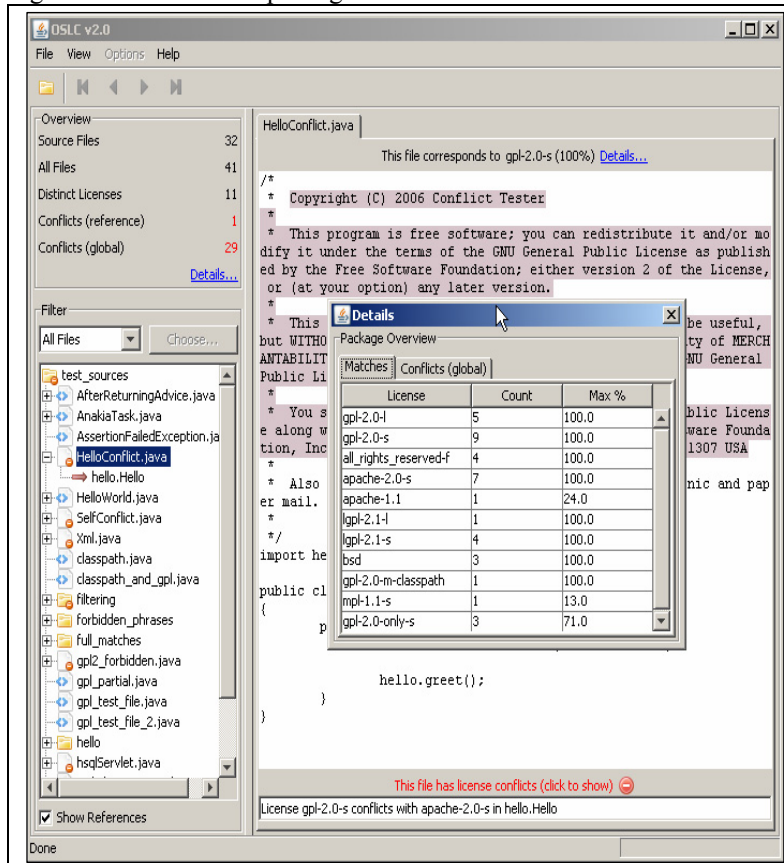
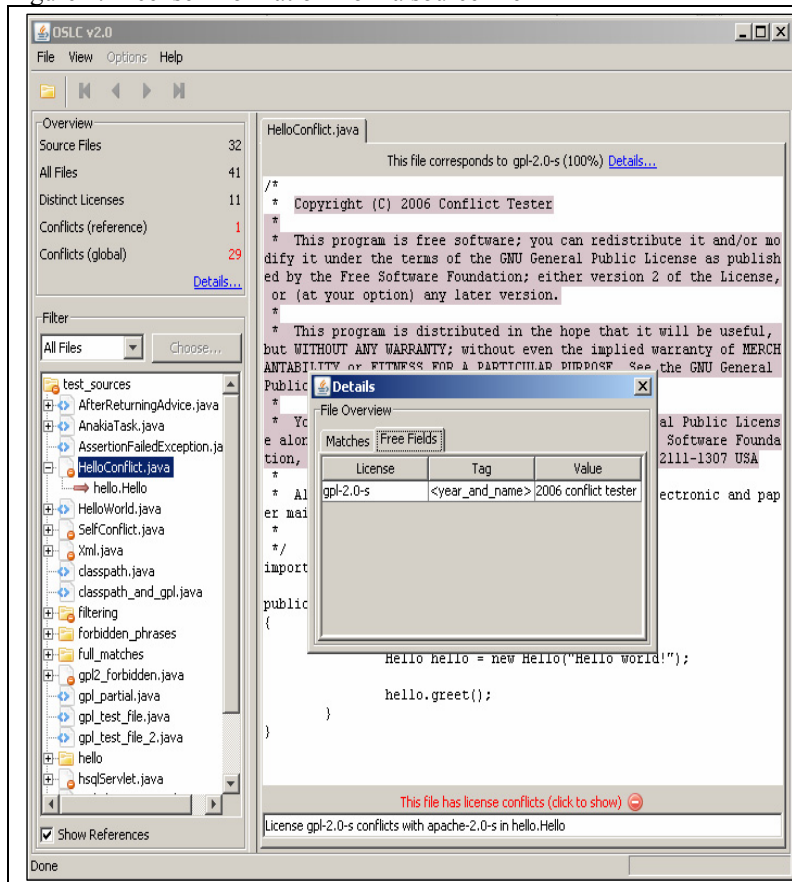


Figure 2: License information from a source file

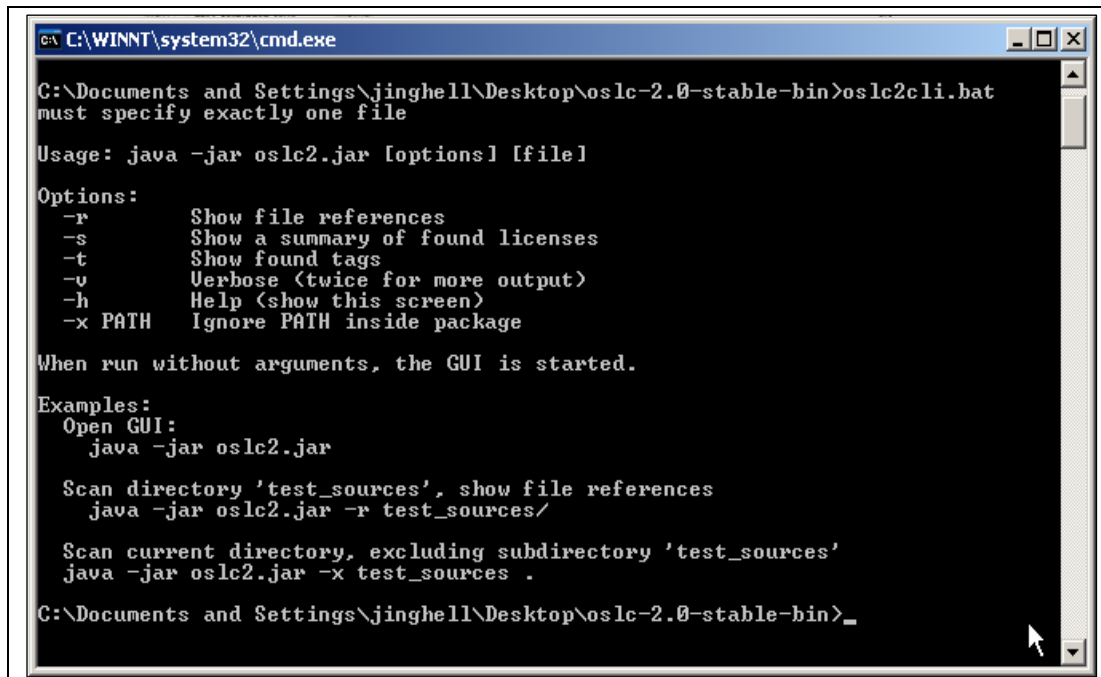


Detailed information about the GUI can be found from the OSLC user manual in the installation package, see OSLC project page in sourceforge.net [9].

3.4.2 Command-line Interface

The command line interface is presented as below:

Figure 3: command line options



```
C:\WINNT\system32\cmd.exe
C:\Documents and Settings\jinghell\Desktop\oslc-2.0-stable-bin>oslc2cli.bat
must specify exactly one file

Usage: java -jar oslc2.jar [options] [file]

Options:
-r      Show file references
-s      Show a summary of found licenses
-t      Show found tags
-v      Verbose (twice for more output)
-h      Help (show this screen)
-x PATH Ignore PATH inside package

When run without arguments, the GUI is started.

Examples:
Open GUI:
  java -jar oslc2.jar

Scan directory 'test_sources', show file references
  java -jar oslc2.jar -r test_sources/

Scan current directory, excluding subdirectory 'test_sources'
  java -jar oslc2.jar -x test_sources .

C:\Documents and Settings\jinghell\Desktop\oslc-2.0-stable-bin>_
```

```
>oslc2cli.bat -s test_sources\full_matches > log.txt
```

```
lgpl_2_1_s.java: lgpl-2.1-s
lgpl_2_1_l.java: lgpl-2.1-l
bsd.java: bsd
apache_2_0_s.java: apache-2.0-s
```

```
Source files:      4
License files:     0
All files:         4
Distinct licenses: 4
Conflicts (ref):   0
Conflicts (global): 2
```

License	Count	Incompatible with
apache-2.0-s	1	lgpl-2.1-l
bsd 1	1	lgpl-2.1-s
lgpl-2.1-l	1	apache-2.0-s
lgpl-2.1-s	1	apache-2.0-s

3.5 Matching algorithm

3.5.1 Overview and list of terms

The most important requirement for this application is reliable identification of licenses. This chapter presents an overview of the license matching algorithm used by the program. Implementation details are not addressed here; please refer to the technical specification of the OSLC program for implementation issues. In this chapter the following terms are used:

Fragment	part of the source or license text defined by the line and character positions of the fragment's start and end point
matching fragment	fragments in both source and license text that have identical alphanumeric characters
Match	a license found in the source text, represented by one or more matching fragments.
true positive match	match that is correct in a sense that the source text actually contains the matched license (or parts of the license)
true negative match	no match is found for a license that is not included in the source text
false positive match	match is found although the source text doesn't actually contains the license
false negative match	no match is found although the source text actually contains the license

Basic goal of the algorithm is to determine if the source text contains text from any of the licenses in the program's license database. The algorithm can be divided in two major parts: fragment detection and match filtering.

3.5.2 Input

Input of the algorithm is the comment text of a source file (plain text, language-specific syntax and has been removed, line numbers and start column positions are retained) and a list of all license texts in the license database.

3.5.3 Fragment detection

For each source text and license text pair a list of matching fragments is needed. Fragments are identified by using a modified version of the algorithm invented by Paul Heckel[10]. This algorithm is similar to the algorithm used in the popular unix program *diff*.

Paul Heckel's algorithm works by identifying unique words (case-insensitive, non-alphanumeric characters are ignored) in both source and license text. After the unique words have been identified, our algorithm uses this data to find matching fragments. This is done by expanding the fragment from a sequential pair of two unique words both backwards and forwards until a difference between source and license text is found. If no unique words are found but there are multiple instances of each unique license template word in the source text, it is assumed that the license text has been duplicated (it appears more than once in the file). In these cases only the first instances of words are taken into account.

3.5.4 Match filtering

Matching fragments found in the fragment detection phase usually contain partial matches from many different licenses. Often the fragments are very small (maybe only a few words) and the fragments can be overlapping (same word in the source text belongs to different fragments in different licenses; note that a given word can't belong to different fragments of the same license). It is clear that many of these matches are false positives. The goal of the match filtering is to remove these false positives while in the same time retaining true positives.

In the current version of the algorithm only the longest fragment and the total length of fragments for each matched license are considered. If length of the longest fragment is below 10% (of the number of words in the license) or if length is below 10 words the match is rejected. This threshold was determined experimentally and it's configurable (higher numbers lead to more positive matches; both true and false).

Overlaps are removed by scanning through the longest fragments in order of fragment length. If fragments overlap, the shorter fragment is cut so that no words overlap (overlap detection is done based on word positions in the source text). If length of the cut fragment falls below 10/10 threshold, the match is rejected. Motivation for overlap detection is that many licenses are simply variations of some other license with long identical segments; without overlap detection there would be many false positive matches.

In the special case of two identical longest fragments that still belong to different licenses; the match with higher total number of matched words is retained. This can happen if there are two licenses with nearly identical license texts.

3.5.5 Output

Results of the matching algorithm are presented in graphical user interface or printed to the standard output. For every file a list of found licenses is reported along with match confidence (percentage of words in the license text found in the longest match fragment) and match position.

The program is able to detect if the original license text has been modified; in these cases the match confidence is below 100% (confidence is rounded down so 100% means exact match with the original license). However the program is not able to detect additions made before or after the original license text unless these additions contain forbidden phrases or parts of some other license.

3.5.6 Advanced matching features

Forbidden phrases

Certain short phrases (such as *Shareware* or *All rights reserved*) that might have legal significance are detected in the source text. A forbidden phrase match is reported only if all words of the phrase are found. A simplified algorithm is used to find forbidden phrases.

Some forbidden phrases are included in certain licenses; if all licenses found in a file contain the identified forbidden phrase it's not reported to the user. Internally forbidden phrases are represented as a special license type.

Modules

Some licenses contain optional additions; for example the GPL license can be combined with Classpath-exception⁷. These exceptions are represented by the program as a special license type; they are matched using the standard algorithm but they are reported only if the parent license is also found in the same file.

Free-form fields

Some licenses contain special fields that can contain any text (such as names of the authors). These field would break the standard matching algorithm if not taken into account. Free-form fields are defined in the license metadata; matching algorithm is able to detect these positions without breaking a match. Any text stored inside free-form fields is saved and presented to the user in the graphical user interface.

There are some restrictions on free-form fields: they cannot be located in the start or in the end of the license (otherwise matching algorithm wouldn't know where the field starts or ends). Also it's not possible to define two free-form field in a row; for example it's not possible to define: <year> <name>, instead <year_and_name> must be used.

Linux kernel support

In the Linux kernel it's possible to specify a license by using MODULE_LICENSE macro. For example it's possible to specify a GPL license by including the following line to the source code: MODULE_LICENSE("GPL"). Because it's not part of the source file's comments, it can't be found by the program's standard approach of scanning only the comment text.

Linux kernel is handled by the program as a special case by using a simple exact matching algorithm to detect these macros. If macro's argument doesn't indicate any known license, it's handled as a forbidden phrase.

Correctness of the matching algorithm

Algorithm's correctness was evaluated by running it against a set of source packages downloaded from SourceForge [9]. Our impression is that the algorithm works well in a sense that it rarely produces false negative matches (assuming that the corresponding license is defined in the program's license database). Also the algorithm doesn't produce many false positives unless a license has many modifications to it's template. When a license has modifications, the algorithm picks the longest fragment while other fragments might be reported as different licenses.

It would be useful to have a qualitative analysis of the algorithm's correctness. For example one could download a large number of random code packages from the sourceforge and measure how often the algorithm performs correctly

⁷ The reference to Classpath-exception: <http://www.gnu.org/software/classpath/license.html>

3.5.7 Future improvements to the matching algorithm

Multi-fragment match filtering

At the moment, only the longest fragment is taken into account when calculating match confidence and when filtering overlapping fragments. All fragments should be taken into account, not just the longest one. This would not be a major modification to the algorithm since all fragments are already detected in the first phase of the algorithm.

Keyword-based matching

As an alternative to the whole-text matching, it should be possible to specify certain keywords (such as license name), that could be used to identify a license. This approach would increase the number of true positive matches in cases when the license text is heavily modified and if the keywords are chosen carefully it should not increase the number of false positive matches. Unfortunately keyword-based matching is unable to detect modifications to the license.

Semi-interactive matching

The user could interactively highlight parts of a source text and mark them as belonging to a specified license. The program would take this information into account and re-evaluate other files. This approach is motivated by observation that license information is often simply copied to different files; in most cases this user intervention would quickly reduce the number of false positive matches found in the source package.

3.6 Architecture

The program has a modular structure; for example support for different source code languages is implemented in the sourceparser module (a java package) that has well-defined public interface. This way it's possible to add support for new languages without extensive modifications to existing code. The modules are:

Module	Main responsibilities
Checker	Logic that connects other modules together. Programming interface for the graphical user interface. Implements the command line interface.
Filepackage	File access from file packages. Implemented for jar,zip,tar and standard file system.
License	Representation of license text and metadata. License database creation from text files.
Matching	Implements the matching algorithm.
Sourceparser	Extraction of comments from a source file and detection of references between source files. Implemented for Java,C++ and PHP.
Gui	Graphical user interface

A repository module was also defined in the initial architecture. It was responsible for accessing source code repositories (such as CVS) but no implementation was coded. Command line interface was planned to be implemented in separate module but at the moment it's located in the checker module.

Basic sequence of actions for the program's standard operation is:

- read the license data and metadata from text files
- present the GUI to the user
- access the source package specified by the user
- extract comments from source files
- run the matching algorithm against source files
- present results to the user

3.7 Future improvements

The program does implement most of the functionality that was planned at the start of the development project. It's possible that development of the program continues in some form in the future. There are many features that could be added to the program, here are some examples (improvements to the matching algorithm have been discussed in chapter 3.4.9).

3.7.1 Support for Creative Commons licenses

Creative Commons [18] is a family of licenses aimed for licensing media content. Different versions of the CC license have different combinations of four basic conditions:

- Attribution: name of the author must be mentioned
- NonCommercial: content can be used only for noncommercial purposes
- No Derivative Works: only verbatim copies are allowed
- ShareAlike: content can be distributed only under the original license

CC licenses can be expressed in three different formats:

- Commons Deed (human-readable code)
- Legal Code (lawyer-readable code)
- Metadata (machine readable code)

CC licenses can also be applied to different file types, for example:

- HTML pages
- RSS feeds
- MP3 or OGG music files
- XMP-enabled documents

At the moment, OSLC program only supports software licenses, but it would be possible to extend the program to include support for creative commons licenses. Since CC licenses can also be expressed as RDF/XML formatted metadata, an XML parser would be needed to read the information. This parser could be implemented as a sourceparser module. Since metadata doesn't contain the actual license text, exact matching algorithm against the license description URL's could be used to identify the

license. Filepackage module implementations would be needed for each of the supported file types.

Since CC licenses are often embedded in web pages, it would be useful if the user could enter an URL instead of a local machine filename. This would require changes to the main program and also a new filepackage module. CC licenses contain multiple language versions; it's not certain if the translations are legally equivalent. If they are not equivalent, they should be represented as completely different licenses. License naming conventions would need to be updated to take into account language versions in a consistent way.

Support for other media licenses (such as the Free Art License) could be added in a similar way.

3.7.2 License database editor

It should be possible to edit the license database with GUI without modifying any text files.

3.7.3 Optimizing random access

At the moment accessing a single file from a compressed package is slow. Random access could be accelerated by using temporary directories or multiple iterators.

3.7.4 Repository support

Support for CVS and Subversion repositories was planned but not implemented. It should be possible to specify relevant repository parameters in the GUI.

3.7.5 Package recursion

Source packages might contain other packages inside the top-level package (such as nested jar files). At the moment only the top-level package is processed.

3.7.6 XML output

The program should be able to output xml for easy integration with external systems.

4. SUMMARY AND DISCUSSION

4.1 Results

In this paper we have presented a brief introduction to the software licensing using open source licenses. Two major types of open source licenses were identified: BSD-style licenses and GPL-style licenses. The main difference between these licenses is that while BSD allows commercial derivations, GPL only allows derivations that are also released under GPL license. Mixing of different license types is problematic; especially when open source code is to be integrated with proprietary code, but also when different open source licenses are mixed together. For these reasons it's very important to be able to identify the licenses that a given software package contains.

We examined two commercial products for managing license information: Black Duck and Palamida. As one of the most popular license compliance management software, Black Duck protexIP presents a comprehensive solution based on Code Print technology. The program compares the source code against millions of commercial and open source code from its KnowledgeBase components. This feature offers a direct and straightforward solution from matching the program code. However, as the matching logic is not available, it is hard to assess the reliability of

the produced result. Palamida contains a rich set of features; especially its license database and the range of alternative scanning methods are impressive. Unfortunately its system requirements are steep and since no demo version or impartial evaluation is available we were not able to confirm its functionality.

We presented our own solution to the license management problem: Open Source Licenses Checker (OSLC). The main feature of OSLC is its ability to identify licenses by comparing source code comments with the license database by using an algorithm derived from Paul Heckel's paper [10]. Other major features are the graphical and command-line user interfaces, ability to open zip, jar, tar and file system source packages and support for Java, C++ and PHP source code languages.

Compared to the commercial products, OSLC has more limited set of functionality but it's usually successful in correctly identifying the licenses unless they are heavily modified from the template license. Possible future improvements to the OSLC program include advances in the matching algorithm and usability improvements.

4.2 Discussion

It seems clear that the importance of license management continues to grow as open source software becomes more widely accepted in commercial environment. Currently there seems to be only two commercial products and one open source solution that deal specifically with this issue; we expect the situation to change in the future as more companies and open source communities become aware of the issue.

Unfortunately it's difficult to obtain reliable information on how well these solutions actually perform in correctly identifying licenses; it would be very interesting to see qualitative scientific research where random source packages (perhaps taken from the SourceForge) would be scanned with these solutions. It would also be interesting to see a rigorous analysis on the algorithms used in these solutions; unfortunately it's unclear if the companies would be willing to expose details of their algorithms, but at least the code of OSLC is available for researchers according to the open source philosophy.

We would be pleased if the OSLC program would continue its life as a vibrant open source project.

References

1. Introduction to IPR: <http://www.piebusiness.info/Intellectual-Property-Rights/Introduction.htm>
2. Juha Laine, 2006, Legal issues in software engineering, <http://tiger.soberit.hut.fi/wiki/lib/exe/fetch.php?media=t763601:legalissues.ppt.pdf>
3. Andrew M. ST. Laurent, 2004, Understanding Open Source Software Licensing (1st edition). O'Reilly Media, Inc.
4. Ville Oksanen, Mikko Välimäki. 2005, License Compliance Software as a Tool for Open Source Risk Management
5. Michael Tiemann, 2006, Approved open source licenses: <http://www.opensource.org/licenses/alphabetical>
6. Harald Welte, 2006, District Court of Frankfurt Issues Verdict on GPL Violation of D-Link, http://gpl-violations.org/news/20060922-dlink-judgement_frankfurt.html
7. Rosen, L. 2004, Open Source Licensing: software Freedom and Intellectual Property Law. Prentice Hall
8. OSSI project homepage: <http://ossi.coss.fi/ossi/>
9. Open Source License Checker 2.0 sourceforge project page: <http://sourceforge.net/projects/oslc/>
10. Paul Heckel, 1978, A technique for isolating differences between files. <http://portal.acm.org/citation.cfm?doid=359460.359467>
11. MIT License Definition: <http://www.bellevuelinux.org/mitlicense.html>
12. BSD License Definition: <http://www.bellevuelinux.org/bsdlicense.html>
13. Free Software Definition: http://www.bellevuelinux.org/free_software.html
14. GNU General Public License: <http://www.opensource.org/licenses/gpl-license.php>
15. Copyleft concepts: http://en.wikipedia.org/wiki/Free_software_license
16. LGPL license: <http://www.opensource.org/licenses/lgpl-license.php>
17. Wikipedia – LGPL license: http://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License
18. Creative Commons definition <http://creativecommons.org/>

APPENDIX 3: Value network analysis of Debian and Eclipse

Analysing structures and operations of open source value networks

Jussi Myllärniemi

jussi.myllarniemi@tut.fi

1. INTRODUCTION

Previously during OSSI –project there have been discussion about network studies on open source (see for example Helander and Laine 2006). The conclusion is that the number of actual network studies carried out from a business perspective on open source is limited. There are some models that take into account the companies' perspectives as the member of network or study the relationships between companies and open source communities. But there are no studies how these models work in practice.

In this value network analysis the purpose is to discover the structure and operations of value networks that are formed in the open source software field. This study concentrates on analyzing two different case communities and five different companies. The analyses of the networks of the case communities help to explain the structures of the existing open source value networks.

Firstly, the concept of open source value network is theoretically introduced. Earlier during the OSSI-project Helander and Laine (2006) introduced some relevant models concerning value networks: ARA (Håkansson & Johanson 1992), which is actually a basic network model, and the Value-Creating Networks (Kothandaraman and Wilson 1999). The main outcome of the theoretical part is the general model of creating and capturing value in open source network that is based on the mentioned theories. Before analyzing the case networks with the help of the following figure, the concept of open source value network is also explained.

Secondly, the value networks of the case communities, Eclipse and Debian, are analyzed. The research data for analyzing the case communities is gathered from different sources; the primary data includes a series of qualitative interviews and a quantitative survey. The quantitative survey is made by Mikkonen et al. (2006a).

Based on the theme interviews, the roles of the companies are analyzed and included in the networks. The analysis of the companies is based on figure X on page XX and the theory presented by Seppänen (2006). Finally, the comparative analysis of the value networks of the case communities summarizes the whole study.

2. THEORETICAL APPROACH

There are many different kinds of actors and roles in open source networks. Together these actors form a large network that consists of lots of different skills. Helander and Laine (2006, p.54) add an interesting point to the discussion about open source developers. They say that the discussion about open source competencies has to be taken down to the level of individual actors, because the competencies of an individual actor play such a remarkable role in open source. Typically, in industrial networks value creation is observed from the viewpoint of the organizations.

Dahlander and Magnusson (2005, p.481) say that the striking feature of open source is that the knowledge needed to generate a software is not controlled by companies. It resides within communities that co-exist with companies. Companies could, though, control the competencies in open source communities by hiring employees to work inside them. Also companies can launch their own communities.

Different kinds of ownerships, questions about resources and relationships, and the roles of customers are hard to figure out in open source networks, and especially when discussing open source value networks. During the OSSI-project previously mentioned theories about value networks offer points of views to observe the value creation in networks, but in the case of open source those general models do not work. Activities, resources and core capabilities are hard to separate from each other, for example, because of the amount of different players the open source environment consists of. In addition, in the open source environment it is hard to say whether the customers benefit from the value or not. Also, because of the amount of players, the value capturing is quite confusing.

The statements mentioned above and the theories presented previously in the OSSI-project were the basis of figure 1 “creating and capturing value in open source network”. The figure takes into account how value is created between different actors in open source network. It also points out who benefits from the value made in the network. The inspiration for the figure was taken from Helander & Laine’s (2006) thoughts. The concept of open source value network is based on figure 1.

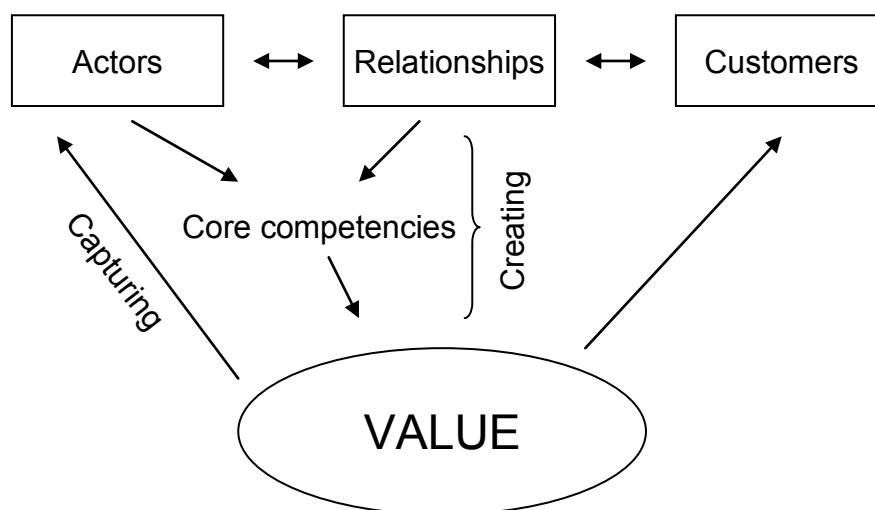


Figure 1. Creating and capturing value in open source network

Actors in the figure mean developers, which are mostly developers in open source communities. Relationships mean the web of companies, which participate in creating value. These relationships are the intermediators between communities and customers. Actually, the same analogy could be seen in the open source value chain made by Räsänen (2004) (see for example Helander and Laine 2006, p.52).

Relationships are needed to connect the needs of the customers and, on the other hand, the needs of the developers. As stated before, they are both end users, and therefore they may both have different expectations of the network and the value it creates. Helander and Laine (2006, p.54) define value as a trade-off between the benefits and the sacrifices the players make in the network. They (ibid.) say that value needs to be created as well as captured by the whole network, not just by the customers.

In the figure the value is created by actors and the companies they have relationships with. Core competencies in the figure represent the resources (skills, knowledge etc.) those players integrate to produce the best value as possible. Together the developers, companies and customers form the open source value network, in which everyone's core competencies are used for value creation. The purpose of the open source value network is to create the value which meet the requirements they asset together.

In the next chapter two different kinds of communities and five different companies are used as examples of analysing open source value networks. The goal of that analysis is to clarify how companies can operate in open source value networks. In chapter 3.4. the analyses of the value networks of the case communities are based on the model presented in the previous figure.

3. EMPIRICAL ANALYSIS OF THE OPEN SOURCE VALUE NETWORKS

3.1. Eclipse network analysis

Eclipse's organization structure is quite hierarchical (see e.g. Eclipse.org), at least from the view point of an outsider. Eclipse is a group of projects and top level projects. Top level projects are managed by the Project Management Committee, which include for example different kinds of councils. Inside one project there are project leads, development teams, subsystems and project plans. Eclipse Foundation is in the top of this whole organisation. (based on The Eclipse Foundation 2007)

The foundation does not employ the open source developers, which are called as committers, but instead the foundation employs a full-time professional staff to provide services to the community. Committers work on the projects of Eclipse; they are trusted individuals who have write access to the source repository. They are typically employed by organisations or are independent volunteer developers. (based on The Eclipse Foundation 2007 and Luoma 2007, p.37)

The committers are mostly from different organisations (a.k.a. companies). The survey (Mikkonen et al. 2006a) conducted in 2006 shows that 60,0 per cent of the developers (or committers in this case) get most of their salary from Eclipse and Eclipse is their main job. 58,7 per cent of the respondents identify themselves

professionally as software engineers, and all of the respondents are highly educated which means they have university education. Most of the respondents get their income from software development (70,5 per cent).

In Eclipse the developers consider themselves closer to the center: 90,9 per cent of the respondents see themselves as a project leader, core member or active developer. This means that the developers do not just develop the code; they also take part in the decision making. It could be said that they have a larger radius of influence. But because there are so many projects running in Eclipse, the proliferation of leader roles is expected, like Mikkonen et al. (2006c, p.26) point out.

As stated before, the developers consider their roles in Eclipse as those who are close to the core. Still, when participating in proprietary software development the answers varied a lot (Mikkonen et al. 2006a). Over 60 per cent have had the previously mentioned roles, but every role had support, for example, bug fixer got almost 14 per cent of the answers. But concerning these questions it is remarkable that developers could choose more than one option. The results of these two questions cannot be directly compared to each other, although there are a remarkable number of answers that are close to the core.

This could be explained by analysing Eclipse developers' attitude towards the companies participating in the open source communities (Mikkonen et al. 2006a). Almost every respondent think that it is good that companies give support to open source projects. The same number of respondents (~95 per cent) thinks that companies' support is harmful, and almost 90 per cent disagree with the claim that companies should not hire employees from open source communities.

Money divides opinions, when the respondents were asked for the reason why they participate in open source projects. Less than 50 per cent say that they participate in open source projects because of money, and almost 30 per cent of answers are neutral. Some of the most interesting conclusions of the survey are the facts that Eclipse developers participate in open source projects because they want to make programs better, they want to learn new skills and they want to share their knowledge and skills.

The survey proves that the developers have face-to-face contact with other Eclipse developers almost every day. There are no studies on how often other communication methods (such as mailing lists, phone, and conversation forums) are used, but in this era those methods could be assumed to be used more often than "traditional" face-to-face conversations. The number of contacts depends on the subgroup (or –task) the developer is working on. The sizes of these groups vary from a couple of persons to hundreds.

Kidane and Gloor (2005) have studied the Eclipse community by analysing open source teams' creativity and productiveness. They studied the 33 Eclipse communities that the whole Eclipse includes, by analyzing mailing lists. Kidane and Gloor (2005) define creativity in this sense of "the amount of feature enhancement carried out by eclipse component development groups". The main conclusion was that the groups that are centralized are found to be less creative when compared to the decentralized ones. The groups that have higher communication density seem to be better performers than those with low density.

Figure 2 takes into account the communication in Eclipse from a larger point of view than for example Kidane and Gloor do. The main purpose of the figure is to clarify how companies are connected and could be connected to Eclipse. This figure pays also attention to companies that are not so strongly linked to Eclipse, in other words are not members of Eclipse. The directions of interaction could also be seen in the figure, although it is not the main point. Of course, the cooperation (or communication) works in two ways, but only the main directions are described in the figure.

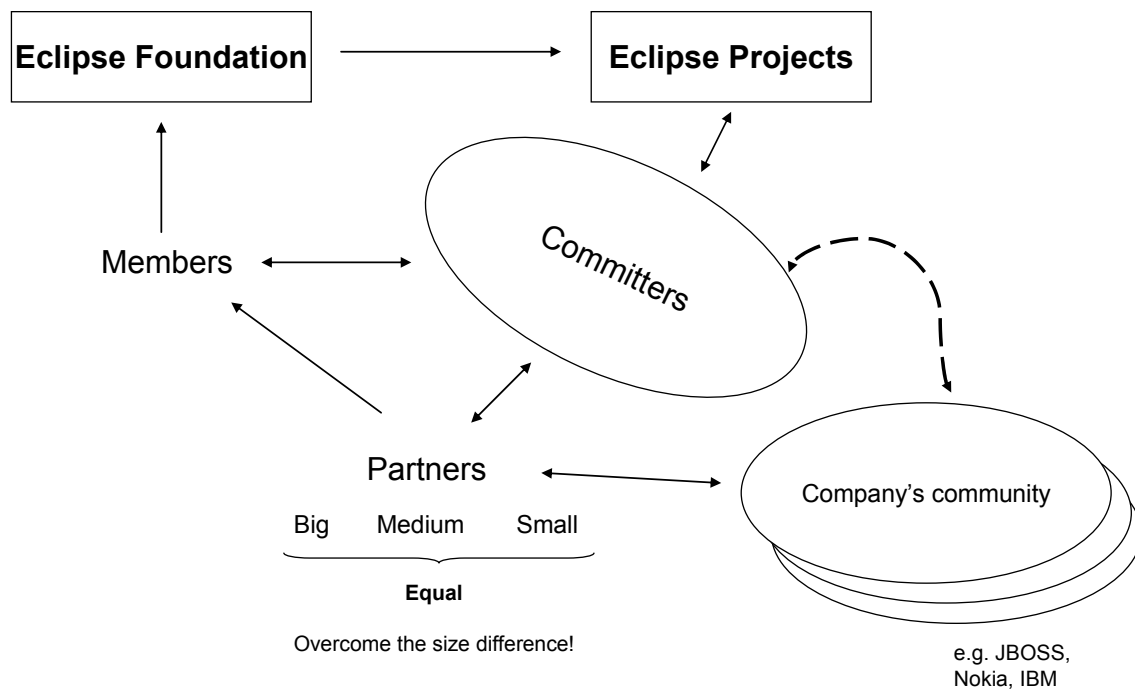


Figure 2. The Partner network of Eclipse

The researcher's analysis, based mostly on www.eclipse.org and Cunningham's (2006) interview, reveals that there are two main ways for companies to benefit from cooperation with Eclipse. The ways mentioned above are to become a member of Eclipse Foundation or to support Eclipse projects (e.g. employing committers to community). And of course, these two ways are not necessarily separate from each other.

Support or cooperation is not so much dependent on the size of your community. Eclipse creates a project environment where a small company could be equal with IBM itself (Cunningham 2006). Cunningham continues that Eclipse evens out the size difference between the partners, and that is where all foundations that care about commercial software should aim at. This is a relevant claim if compared to what has been stated previously about the communities. In the open source value network value is not only created to potential customers, but also the developers of the communities are the end users. Overcoming the size difference is something that fits to the ideology of a traditional open source developer.

Another way to cooperate with Eclipse is through the company's own community. For successful companies such as IBM, Nokia and JBOSS, which Cunningham particularly mentioned, it is common that they have their own communities. For example, Nokia works closely within communities to develop software; their engineers take part in the community work. So far, it is important to note that through the communities of companies, the companies could offer more completed products to committers for further development. The committers could also be used as testers etc.

3.2. Debian network analysis

Debian has surprisingly clear organizational structure although it is a community based on voluntariness. While Debian is the oldest community under investigation, it is also the largest community, if measured in developers. It had almost 1000 voting members in 2005, but there could be more: the Debian mailing list had over 2000 members when the survey took place, so it could be said that there are 2000 maintainers. (based on SPI 2007)

The survey shows that approximately 50 per cent of the respondents consider Debian as a hobby, but still almost the same number of developers (51,2 per cent) have a full-time job. 25,6 per cent are full-time students. But professionally they identify themselves more versatile than for example Eclipse's developers: almost 24 per cent are software engineers, 12,9 per cent are consultants and the rest are divided into students, programmers etc. The developers are not as highly educated as in Eclipse. Approximately 70 per cent have a university education, but in addition, about 20% of the respondents are highly graduates.

The roles of Debian developers, according to the survey, are not so close to the center as the roles in Eclipse. Most developers (nearly 80 per cent) think they are active or peripheral developers. The active developers regularly contribute new features and fix bugs, while peripherals contribute only occasionally. This fact strengthens the view that participating in Debian is more of a hobby than a profession for the developers. But because the sample in the survey was quite small, one cannot make completely reliable conclusions. In the case of Debian the returning rate was 4,2 per cent, while the Debian's list had 2024 subscribers and 83 respondents. This could be explained by the number of active developers which could be smaller than the number of subscribers, as Mikkonen et al. (2006b) explain.

Like in the case of Eclipse, also the roles of Debian developers differ in the community and in proprietary software development. Almost 35 per cent have been project leaders and core members in a proprietary software development, while only less than 5 per cent have these roles in the community. This could be explained by the fact that companies appreciate the Debian developers more than they imagine. The developer's knowledge has been found useful in the managerial duties of proprietary software development. Also one fact is that Debian is just a hobby for developers while they get their salary from somewhere else.

The attitude of the Debian developers toward companies' participation in the projects of the communities is comparable to the attitude of the Eclipse developers. It is good that private companies give support to open source projects (94,1 per cent). Over 90

per cent of the respondents disagree with the claim that the support is harmful, and they are not against the fact that companies hire employees from open source communities (almost 90 per cent). But it should be recognized that in the case of Debian the support from companies is more like donations if compared to Eclipse. Still, Debian has partner programs, which include the roles of development or service partner.

Though Debian has this partner program, and it has many huge companies as partners (e.g. HP, Sun, Simtec Electronics), Wirzenius (2007) names the upstream developers and the users as the main cooperative partners of Debian. The upstream developers are the main developers, who are expected to fix bugs and maintain as well as develop their software. Still according to Wirzenius (2007), Debian expects to receive reports from the users concerning problems.

According to SPI (2007), Debian works close to its partners for ensuring that it understands the needs and concerns of the partners, and vice versa. Debian expects, for example, promoting and advertising from the partnership. And as a compensation for partnership, Debian recognizes partners officially and maintains a good working relationship with them. Partners could also be as donators.

The survey proves that Debian developers have no face-to-face contacts with each other. Only 27 per cent of the respondents had face-to-face contacts more often than once a month. This fact is supported by the official websites of Debian; the communication is mainly done through e-mail and irc (SPI 2007). The number of contacts depends on the subgroup (or –task) the developer is working on. The sizes of these groups vary from a couple of persons to hundreds, like in Eclipse. 50 per cent of the respondents have contacts because they work on the same subtask, and almost 24 per cent have contacts because they are friends.

The next figure describes the directions of interaction, which the arrows symbolize. Figure 3 presents the Debian community and the main players around it. The responsible roles mentioned above (e.g. the Debian Project Leader, The Technical Committee) are included in the developer community and in the upstream developer teams. The upstream developer teams are the “cell” which is a more important cooperation partner to Debian than others.

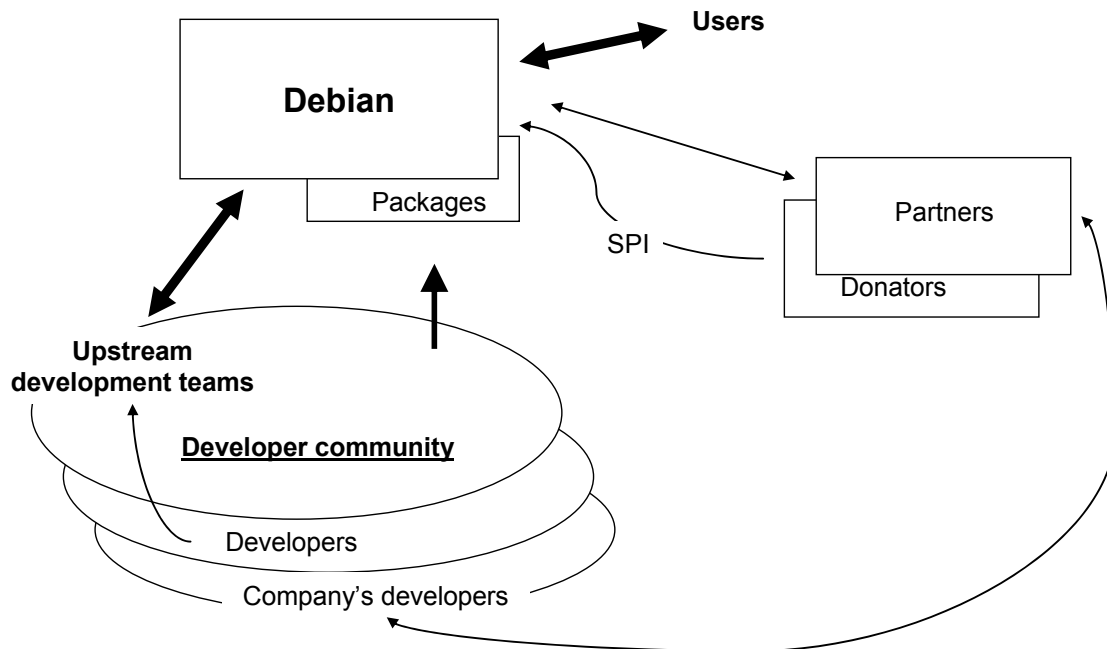


Figure 3. Direction of interaction between major players in Debian.

The same analogy that presented in the figure could be seen in Kothandaraman and Wilson's (2001) figure "Model of value-creating networks (see Helander and Laine 2006, page 50). Analogy means the interaction between the core "players". Although the figure is for understanding the value creating process and its links to the core capabilities of the firms in the network, it could also be used in the case of Debian.

The main software development is done by upstream developer teams. These teams could be seen as the main competencies of Debian. The value it produces is created by these teams. The value strengthens the relationships between Debian and the developers. It also strengthens the relationships between Debian and other players. For example, users who, for example, report from bugs, define the relationship to Debian according to the kind of value or benefits they get from it.

The users are separated from the developers in this figure. These users mean users outside the community. The reason they are separated is because Wirzenius (2007) particularly mentioned those as important partners.

The arrows in the figure reflect the way companies can affect the cooperation with Debian. One way is through a developer community by employing some developers of companies to work with Debian. Upstream developer teams are in an important position in the decision-making, and their work for packages is remarkable, therefore it is important for the companies to get their employees in those kind of positions. Contacts with the upstream developer teams may be considered to be the most important contacts from the point of view of the partners (and the companies).

The support the partners give is compensated by the value they get. It is also very common that companies' own developers are a part of the Debian community. Consequently, the support the companies give is also compensated through community to Debian, as presented in figure XX. The support the companies give can be donations. SPI is the way the donators support Debian. SPI (Software in the Public

Interest) is a non-profit organization formed to help other organizations create and distribute free/open-source software and open source hardware. Debian uses it for handling money donations. In this case the cooperation could be considered to occur in only one direction rather than interactively as in partnering.

3.3. The analysis of the case companies

As stated before, the company analysis is based on the interviews made during the spring 2007. In addition to the interviews, the material was gathered from the internet and from other secondary sources. The companies under analysis were F-Secure, IBM, Nokia Networks, Novell and Plenware. The following figure illustrates the OSSI framework where the case companies are added. The position of the companies is based on the analysis of the interviews.

F-Secure utilizes open source software applications, and, at some level, open source software could be seen as a tool in research and development. F-Secure is interested in communities, but not enough to be considered as actively participating in the management of the communities etc. Actually, the company is interested in improving knowledge of solutions that are not created only for open source customers but also for the developers of open source communities. It could be said that the company is not very interested in utilising traditional open source but it is interested in how they can integrate their services to “community –thinking” and to open source environment as a whole.

If compared to figure 4 F-Secure does not contribute open source in a way it is analyzed from the point of view of sociology. In technology and business aspects it is positioned to the middle of OSS application utilizers and the type which uses OSS tools in research and development.

Like F-Secure Plenware utilizes open source software both for technology and business benefits. The company uses open source tools in software development, utilizes open source application platforms and databases, and integrates OSS components for customer solutions.

In the same level with Plenware is IBM business unit. Actually, because of the size of IBM, there are different approaches for IBM corporation and for the interviewee’s Finnish business unit concerning open source. The corporation sees open source as an opportunity for growth; utilizing open source expands the markets for information technology services. A business unit’s open source strategy could be considered from an economical and innovative viewpoint. It is a tool for cost efficiency, it offers competitive and price advantage. The business unit of IBM does not invest in open source, it utilizes open source applications. The unit benefits from the work the communities do. But the corporation invests, for example, money to communities and in exchange they get some software development (information, technology etc) from communities.

As the whole IBM, also Nokia works in every level of the sociology and technology aspects. Nokia does not launch communities but they work on all the other levels in the business aspect. This means that Nokia does not launch any open communities for outsiders. The company utilize open source software as much as possible because the

main purpose is to produce value to customers, and Nokia understands the possibilities the open source can offer. Flexibility, speed and cost savings are the reasons for using open source.

Another company that contributes open source in every level is Novell. The company has used open source components for years in some of Novell's own products; components from MySQL, JBoss etc. Actually, Novell develops open source software as a part of their business, which is quite unique in the business world. The company has tried in every possible way to be a part in communities and to support the idea that open source is a central way of distributing and developing programs. In general, this can also be considered from Novell's investments concerning open source: they have released their products, maintained the conversation of open source in many ways etc.

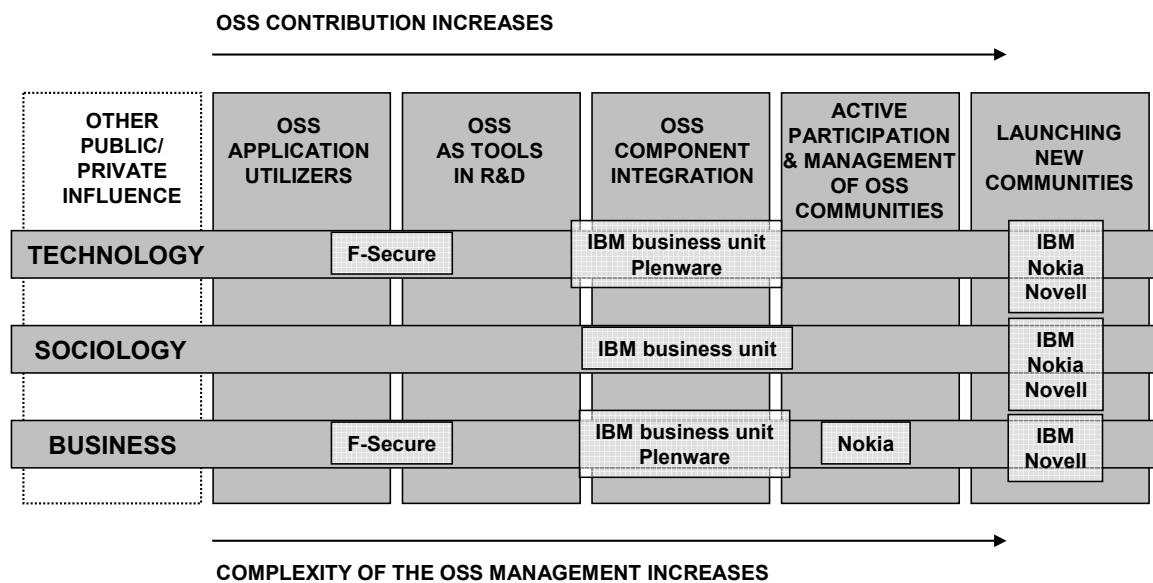


Figure 4. Position of companies in the OSSI framework

In addition, table 1 introduces another summary of the company analyses.

F-Secure is more of a follower when it comes to open source. This means, for example, participating in research projects. Comparing F-Secure with the role aspect in the figure 4 their type of involvement is on the level of observer or user. It is hard to place F-Secure to a certain role because of the information presented in this chapter. The observer follows open source development, but user uses the applications that it finds more valuable to its business. In the case of F-Secure, open source is not the main thing because they are interested in communities.

Opposite to F-Secure, Plenware has formed a strategy concerning open source; it is seen as a clearly strategic issue. They have for previously used open source components and platforms in several projects, but during the last few years they have invested more in open source. Therefore, Plenware is classified as a user or an adapter, or an integrator. From the integration point of view, Plenware's own VoIP system using OSS Asterisk PBX is a good public example. Internal studies at Plenware have found among personnel a positive attitude towards open source and a high level of voluntary participation in OSS projects. Still, it is important to notice

that Plenware's main business is providing software development services for top companies and that is why the company is difficult to position as a single user type.

On the other hand, the positioning of the corporation of IBM and the interviewee's business unit is quite easy. The Finnish business unit works as an application utilizer or component integrator, which could be concluded from above. The user type for the business unit is also the integrator, while the corporation is the promoter. It is typical for promoters to invest in communities, in the case of IBM investing means money and patents. But in this case the business unit benefits, while the corporation invests. At least this is the viewpoint of IBM and for example in the case of Nokia, the roles are the other way round.

Like in the case of IBM also Nokia is hard to handle as the whole corporation, and that is why Nokia is analyzed from the viewpoints of the whole corporation (as Nokia) and Nokia Multimedia. Nokia is the user or adapter if considered from the points of view of the user types (cf. table 1). Nokia uses browsers which are based on open source software. The unit of the interviewee, that is a part of Nokia Multimedia, could be seen as the integrator, because it uses the Linux operating system, or the engine, because Gnome is an important partner, or the promoter, because it has developed the maemo.org.

Like said before, there are no open source software user types that Novell does not fill. For example, Novell uses lots of different wikis and other social communication tools. For example, Novell uses wikis in documentation. In addition, Novell has its own internal developing community. Besides maintaining its own community, Novell works closely with communities, which is typical for a promoter. According to the interviewee, besides Red Hat and IBM, Novell has most open source developers in different kinds of community projects in the world.

Table 1. Involvement types of the case companies

Type of involvement	Primary target	Sacrifices	Benefits	Community
Observer	Keeps distance, follows development	Does not get involved	No investments, timing advantage	May follow discussions
User	Picks raisins from the bunch	Cannot guide development	Benefits others	May follow discussions
Adapter	Efficient	Cannot guide development	Cannot guide development toward own interests	Has a weak link
Integrator	Integrates with its own development	Possible image lost		Typically has stronger connection
Engine	Leads development	Investments	Gets own ecosystem	Has committed oneself to certain community
Promoter	Brand benefits with engine	Large investments	Own brand and image	Maintains and coordinates community

Dahlander and Magnusson (2005, p.482) state that there are numerous factors which explain the differences in the performance of companies dealing with open source.

One is that certain firms just have superior capabilities, they have superior products, or they are better in their exploitation activities than others. But one reason might also be that some companies have better relationships to open source communities.

In the next chapter the roles mentioned above are connected to the value networks of the case communities. These roles are mostly discussed on a general level without going into any details of a certain company's characteristics. The chapter also takes a stand on the relationships formed between communities and companies.

4. THE VALUE NETWORKS OF THE CASE COMMUNITIES

Previously in chapters 3.2 and 3.3 were presented the communities of Eclipse and Debian. Some of the most important facts concerning the structures of both communities were presented. On the other hand, the chapters do not take into account very deeply how value is created in the networks. In the following figures 5 and 6, the value networks of these two open source communities are presented. These figures take into account how value is created, what the main competencies of these communities are, and who benefits from the created value. After the discussion on value creation, the relationships between the communities and companies are taken under deeper investigation in chapter 3.6. The chapter also discusses the different ways to operate in the open source value networks.

4.1. Differences of the case networks

Figure 5 presents the value network of Eclipse. Eclipse is more business oriented than Debian, which can be seen in the figure, too. The main players are the committers of Eclipse, the partner companies and their communities, and the customers.

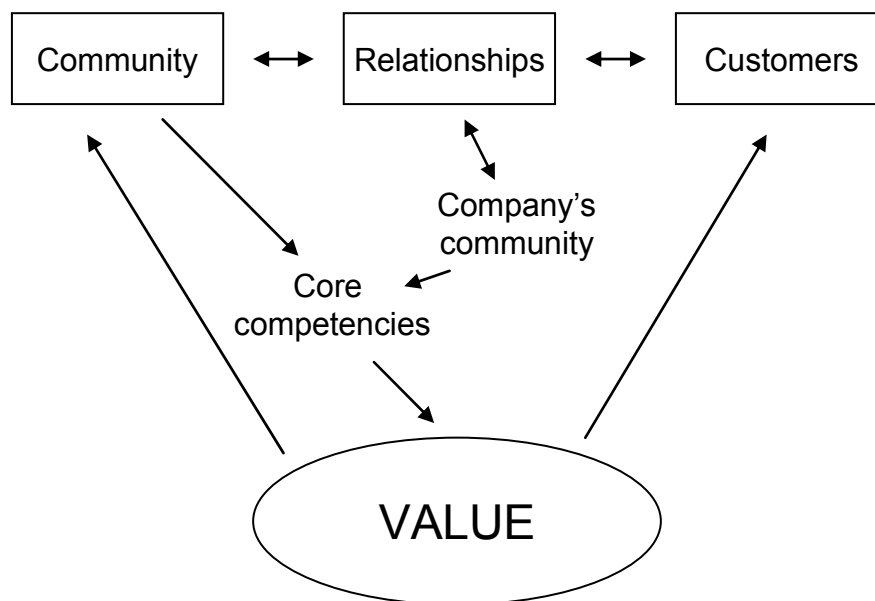


Figure 5. Value network of Eclipse

The value network of Eclipse has same characteristics as the model of creating and capturing value in open source network which was made from the perspective of the companies. The models are similar because Eclipse acts very professionally as a community, which has been proved already in this study. Only the community and the company's community are different. It is also important to note that there are no previous mentions about the customers in the network of Eclipse but they were added to this figure because of the value network of Eclipse is so business oriented and in business networks the companies always take those into account.

The community includes the developers of Eclipse. Those developers are typically employees from some organizations or independent developers. They work closely with the companies. The relationships represent the web of partners, members and other companies. Although the member organizations of Eclipse invest a lot in Eclipse, they are not mentioned separately in this figure. This is because Eclipse evens out the size difference between the companies, and so all companies have the same possibilities to affect Eclipse.

The purpose of the company's community is to emphasize a way how companies can interact with Eclipse. Although there are no studies on how these communities work with Eclipse, it is clear that successful companies have their own communities. According to the survey made in 2006, over 80 per cent of the developers of Eclipse that participated in the survey thought that companies should employ their own developers to open source projects. Therefore, it could be concluded that if the company has its own community, the relationship to Eclipse could be more efficient.

Together the community of Eclipse and the companies form the core competencies which create the value of the network. The conclusion that people are the main competence in communities and, in this case in Eclipse, could be drawn based on several sources (e.g. the Survey, and Goldman & Gabriel 2005). Thus, the willingness of the developers of Eclipse and also the developers of Debian, to develop themselves, to help each other and to share their knowledge is crucial.

Although the people in both Eclipse and Debian are willing to share their knowledge and to help each other, it is strange that the communication in their case is totally different. There are no face-to-face contacts between Debian developers. Of course, one explanation is that Debian is more like hobby to developers, and the communication is handled besides other tasks.

According to the survey over 85 per cent of the respondents of Debian developers thought that companies should employ their developers to open source projects. Also according to the websites of Debian, the community aims to work in a close relationship with its partners. The partners are highly appreciated. Actually, figure 6, where the value network of Debian is described, takes into account these facts. The main players in this figure are the developer community of Eclipse, the relationships, the users, the customers and the upstream developer teams.

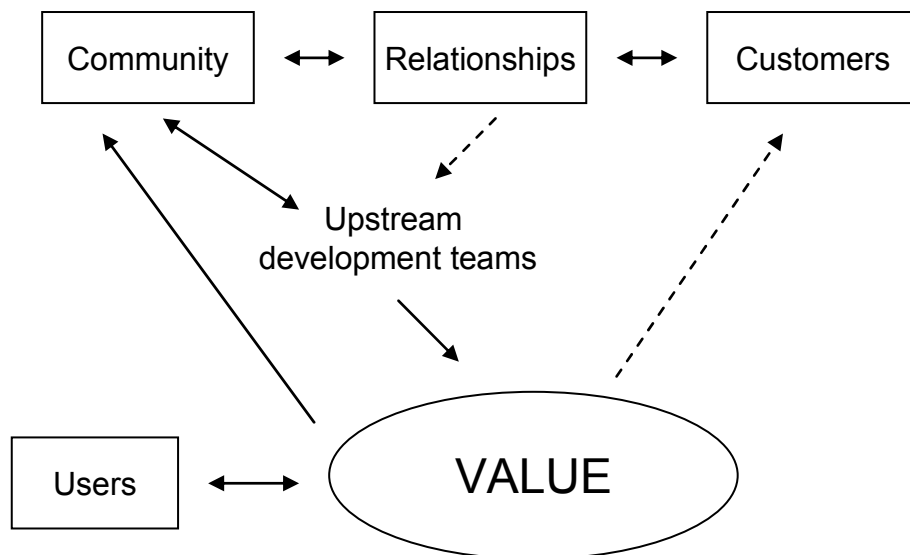


Figure 6. Value network of Debian

Like in the case of Eclipse also in this figure the relationships are formed between the partner companies and other companies which are interested in it. The relationships also include donators, which Debian appreciates. For example, the survey proofs that the developers of Debian thought that donations are a useful way for companies to collaborate with Debian.

The arrows from relationships to upstream developer teams, and also from value to customers are dashed because of the uncertainty of the roles of the companies and the customers in the value creation of the value network. In chapter 3.2 it is stated that the upstream developer team forms the core competency of Debian and the teams are the main actors who create the value. The companies interested in Debian should be emphasized directly, not just through the community. There is no certainty if companies already do this, but the results of the survey and the interview of Wirzenius do not support this statement very strongly.

Like in the value network of Eclipse, Debian does not create value to the customers, at least not so visibly. Because Debian is based on voluntariness, the developers create value mainly to themselves and to the end users. In figure 6 the users are separated from the customers and community developers because of their importance to Debian. Debian does not work as professionally as Eclipse, and that is why its end users are not the same ones with customers.

Based on the fact that the Eclipse developers have larger influence on the community and they take part in the decision-making, and reciprocally the Debian developers base their interest towards their community on voluntariness, one could draw a conclusion that approaching the developers of Eclipse is more certain or safer from the business perspective.

As mentioned before, one reason for Eclipse being more business oriented than Debian, is their different ideologies. Approximately 60 per cent of Debian developers said that they develop software because it should be free. On the other hand, over 60 per cent of Eclipse developers disagree with the claim. It seems that the developers of

Debian follow the same ideology the free software was based on, though like argued above there are facts that support doing software business more professionally in Debian.

Dahlander and Magnussen (2005, p.489) noticed that norms and values cause challenges between communities and companies. Actually, they (2005, pp.489-490) noticed that there are some managerial issues that are critical to attend to in relation to the community from the perspective of the company. Their challenges are easily linked to this thesis, because in the case of each challenge there are similarities to the analysis of this study. They (ibid.) base their study on observing the case studies of Nordic open source companies. More about these challenges and in general, the ways to operate in open source value networks are in the following chapter.

4.2. Challenges of operating in open source value networks

One challenge is about the value and norms which were mentioned already. Firms that have key individuals within projects have the possibility to handle the boundaries between communities and companies. Also Dahlander and Magnusson (2005, p. 489) mention that companies that have established communities have greater influence to communities. O'Mahoney and Ferraro (2004) (in source: Dahlander & Magnusson 2005, p.489) emphasize the importance of face-to-face interactions in managing the boundaries of open source.

Some of the case companies, which are called promoters, have their own communities or they maintain and contribute to some communities. But the company does not have to be the promoter or engine type of the company to overcome this challenge. The integrator or even the adapter could guide the development of communities, if not by itself, at least in cooperation with its partners.

Another challenge is handling the different licenses. Although the license issues have not been studied very deeply in this thesis, they are important because the licenses affect the ownerships of open source projects and also have symbolic value (modified from Dahlander & Magnusson 2005, p.489). According to the survey made by Mikkonen et al. (2006), Eclipse developers prefer CPL (almost 60 per cent) while Debian developers almost unanimously prefer GPL or LGPL (over 80 per cent). Companies which are used to develop commercial software might find it hard to use the open source licenses, especially GPL.

Licenses are one solution for the control and ownership issues. Dahlander and Magnusson (2005, p.490) point out the business model MySQL chose to resolve this problem. MySQL used dual licensing to make a difference between paying users and non paying users. It could be said that licenses, and from a broader perspective, business models, direct the relationships with companies and communities. For example, IBM uses a patronage model with Eclipse.

On the other hand, Debian developers prefer GPL, and also the fact that Debian is more like a free community, supports the use of certain models and licenses with it. But still, it is hard or almost impossible to say that a certain type of company (cf. the roles on table 1) must use a certain business model when dealing with the case companies. Plenware is a good example. The company is identified as a user or an

adapter, but with some partners it develops communities to certain directions and the role of integrator could be used. The roles, in general, are dependent of the business model and, therefore of, the license the company uses.

Nowadays although there are communities like Debian, the contributions the companies make to open source assist to form communities like Eclipse. Open source is moving towards a more professional way of doing business, and more and more actors appear to the open source networks. This means that the existing companies in the open source networks must improve their relationships to the communities they deal with.

So far value is created, for example in Debian, to developers and users, while in Eclipse it is done more strongly to customers. One challenge that also Dahlander and Magnusson (2005, p.489) have figured out, is the different interests when it comes to the nature of the work between companies and communities. Companies want to create value to customers while communities prefer to create it to themselves. Actually, this is something that has already been discussed in this chapter when the reasons behind the structure of the value network of Debian (cf. the dashed lines in the figure 3) were considered.

The case companies do not have any specific conflicts with the communities. Even though some misunderstandings exist on the ideological level, there are no conflicts that could be connected to a certain role that were presented in table 1. Two of the challenges Dahlander and Magnusson (2005) present are related to the previous statement. One is about control and ownership that occur especially with the firms that are active in creating new projects, and another is about avoiding direct conflicts with the communities (modified Dahlander & Magnusson 2005, p.490). When considering the conflicts within communities it should always be noticed that communities consist of thousands of people who all make up their minds independently. The most popular communities cannot have only supporters, as the CEO of MySQL Mårten Mickos (2007) states it.

The interviews show that there are problems inside the companies neither. There was no user type that has problems with the activity of the developers towards open source. Actually, some of the case firms were surprised by the voluntariness of their employees. This is actually a contradiction to Dahlander and Magnusson's (2005, p.489) study, which claims that one challenge is to attract developers to contribute and users to use the software or the product.

This might be a challenge when employing outsiders from communities for the development, but when employing own developers to work with a certain community there should not be problems. Though, the open source environment offers so many and so different kinds of programs and techniques to developers and users which compete with the company's own methods that this challenge is not so hard to imagine.

In value creation, resources are essential. In the previous chapter the employees were presented as one resource that companies could use to affect the communities. Resource consumption related to community development is a challenge mentioned by Dahlander and Magnusson (2005, p.489). This is something that the promoters of

this study have noticed as well the investments to open source and to communities must be significant. Besides time and money, those types of companies have released patents and given up copyrights etc. In return for the investments the promoters capture the benefits from value creating. But large investments are not the only way to cooperate with communities. For example, donations to Debian are an easy way to affect the development done in Debian and by donations the jump to partnering is not so long. Debian has admitted that partners are highly valued.

5. SUMMARY

In open source value networks, value creation is usually centered to some communities. In the case networks value creation is different because the values and norms of the participants differ from one another. Because of that, Eclipse community seems to be most appropriate for companies who are willing to develop the code in the sense of making money. Debian, on the other hand, which is a much larger community than Eclipse, is closer to the basic ideology of the whole open source concept and, therefore, regulates the development of the open source movement in general.

In the case of Eclipse, the customers play a more important role than in Debian. This is supported by the statement that in the network of Eclipse the value is created to the customers, while in the network of Debian the value is created to the developers and the users. This is partly because of the different values and norms the developers have in the case communities.

In both case communities, the developers are one important part of the core competence that contributes to the value. Diverse backgrounds and skills are the richness of the communities. This is what all the participants should understand. No company can lead the communities by itself in open source value networks because there are too many players involved. Together the developers, companies and also customers form the open source value network, and they set the requirements for the value which they create together.

The developers and the companies form the core competence in the value network of Eclipse. This emphasizes the importance of the relationships between communities and companies, in general. The study shows that the promoters get the best benefit out of open source because they have made the largest investments. They have, for example, launched their own communities. By emphasizing the relationships between the own communities of the companies and the communities (like Eclipse and Debian) the best benefit can be achieved. In the case of Debian, the upstream developer teams are the core competence that every type of companies should be connected to. By interacting with the communities, the companies could more efficiently follow the discussion concerning open source, and at the same time share the knowledge of the firm. Actually, the survey proves that the developers of both case communities are willing to share and adapt the knowledge. This is something the firms that are interested in open source should consider, for example, when the companies want to increase the awareness of the company among the communities.

REFERENCES

Dahlander, L. & Magnusson, M.G. 2005. Relationships between open source software companies and communities: Observations from Nordic firms. *Research Policy*. Vol 34 (4). pp. 481-493.

Goldman, R. & Gabriel R. P. 2005. *Innovations Happens Elsewhere - Open Source as Business Strategy*. The United States of America, Morgan Kaufmann. 402 p.

Helander, N. & Laine, J. 2006. The Value Network Approach to Open Source Software Business. In: Helander, N. & Martin-Vahvanen, H. (eds.) 2006. *Multidisciplinary Views to Open Source Software Business*. Tampere, eBRC Research Reports, 33. pp. 46-57.

Håkansson, H. & Johanson, J. 1992. A Model of Industrial Networks. In: Axelsson, B. & Easton, G. (eds.) 1992. *Industrial Networks. A New View of Reality*. London, Routledge. pp. 28-36.

Kothandaraman, P & Wilson, D. T. 2001. The Future of Competition: Value-Creating Networks. *Industrial Marketing Management* 30/2001. pp. 379-389.

OSSI Research Group. 2007. OSSI Research Project – Managing Open Source Software as an Integrated Part of Business. [http://ossi.coss.fi/ossi/fileadmin/user_upload/Other/ossi_seminaari31052007_final.pdf]. Read 2.7.2007

Räsänen, P. 2004. Opening Speech at Open Mind Conference 11.11.2004.

Seppänen, M. 2006. Thoughts on competitive strategy and OS. In: Helander, N. & Mäntymäki, M. (eds.) 2006. *Empirical Insights on Open Source Business*. Tampere, eBRC Research Reports, 34. pp. 4-10.

Empirical Material:

a) Qualitative interviews

- Company interviews:
 - F-Secure / Janne Järvinen: 5.4.2007
 - IBM / Juha Hulkkonen: 30.3.2007
 - Nokia Multimedia / Ari Jaaksi: 10.4.2007
 - Novell Finland / Kim Aaltonen: 5.4.2007
 - Plenware / Pauli Kuosmanen: 11.4.2007
- Community interviews:
 - Debian / Lars Wirzenius: between 23.-25.5.2007
 - Eclipse / Ward Cunningham: 17.5.2006
 - MySQL / Mårten Mickos: between 9.-19.4.2007

b) Quantitative data

- Mikkonen, T., Vadén, T.& Vainio, N. 2006a. FOSS survey.

c) Secondary data

- Kidane, Y.H. & Gloor, P.A. Correlating Temporal Communication Patterns of the Eclipse Open Source. [http://www.casos.cs.cmu.edu/events/conferences/2005/2005_proceedings/Kidane.pdf]. Read 5.6.2007.
- Luoma, I. 2007. On Software Engineering in Open Source Software: A survey of selected projects. Master of Science Thesis. Tampere, Tampere University of Technology, Department of Information Technology.
- Mikkonen, T., Vadén, T.& Vainio, N. 2006b. Survey on four OSS communities: description, analysis and typology. In: Helander, N. & Mäntymäki, M. (eds.) 2006. Empirical Insight on Open Source Software Business. Tampere, eBRC Research Reports, 34. pp. 52-66.
- Mikkonen, T., Vadén, T.& Vainio, N. 2006c. Open Source Communities: A Mix of New, Old and Very Old Characteristics. In: Helander, N. & Antikainen, M. (eds.) 2006. Essays on OSS Practices and Sustainability. Tampere, eBRC Research Reports, 36. pp. 15-31.
- SPI. 24.5.2007a. About Debian. [<http://www.debian.org/intro/about>]. Read 5.6.2007
- The Eclipse Foundation. 2007. About the Eclipse Foundation. [<http://www.eclipse.org/org>]. Read 15.3.2007

APPENDIX 4: OSSI research in a nutshell

OSSI Research Project Publications by themes

	OSS In general	Debian	Eclipse	GNOME	MySQL	Laika
Technology	19					1
Sociology	4, 22					
Law	6,7,8,13,14	17, 18				
Business	2, 3, 9, 13, 16, 22, 25	25	25			1

Refereed book chapters

1. Järvensivu, Juha., Helander, Nina. & Mikkonen, Tommi. Dependencies, Networks and Priorities in an Open Source Project. In Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives. (Eds.) Kirk St. Amant and Brian Still, Texas Tech University. Ch 10.
2. Seppänen Marko, Helander Nina & Mäkinen Saku. 2007. Business models in OSS value creation In Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives. (Eds.) Kirk St. Amant and Brian Still, Texas Tech University. Chapter 45.
3. Puhakka Mikko, Jungman Hannu & Seppänen Marko. 2007. Investing in Open Source Software Companies: Deal Making from a Venture Capitalist's Perspective. In Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives. (Eds.) Kirk St. Amant and Brian Still, Texas Tech University. Chapter 41.
4. Vainio, Niklas. & Vadén, Tere. Free Software Philosophy and Open Source. In Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives. (Eds.) Kirk St. Amant and Brian Still, Texas Tech University. Ch 1.

Journal Articles

5. Mikkonen, Teemu., Vadén, Tere. & Vainio, Niklas. 2007. The Protestant ethic strikes back: Open source developers and the ethic of capitalism. First Monday, volume 12, number 2 (February 2007).
6. Oksanen, Ville. & Välimäki, Mikko. 2006. Free Software and Copyright Enforcement - A Tool for Global Copyright Policy? Knowledge, Technology & Policy Winter, Volume 18, Issue 4, pp. 101-112.

7. Välimäki, Mikko. 2005. Software Interoperability and Intellectual Property Policy in Europe. *European Review on Political Technologies*. Dec. 2005.

8. Välimäki, Mikko. 2006. Copyleft Licensing and EC Competition Law, [European Competition Law Review](#) 3/2006 (Volume 27, Issue 3), pp. 130-136.

Articles in Conference Proceedings'

9. Helander, N. & Rissanen, T. 2006. Value-creating Networks Approach to Open Source Software Business Models. In Seppä, M. Hannula, M. Järvelin, A-M, Kujala, J. Ruohonen, M. & Tiainen, T. (eds.). *Frontiers of e-Business Research 2005*. Tampere University of Technology & University of Tampere.

10. Järvensivu J., Helander N. & Mikkonen T. The network characteristics of Open Source software business: a multi-disciplinary case study. The ICEB & EBRF Conference Tampere, November 28- December 2, 2006.

11. Järvensivu, J., Kosola, M., Kuusipalo, M., Reijula, P. and Mikkonen, T. Developing an Open Source Integrated Development Environment for a Mobile Device. International Conference on Software Engineering Advances, Tahiti, French Polynesia, Oct. 29.-Nov.3., 2006.

12. Kujala J., Helander N. & Lehtimäki H. Analysing Multi-voiced Strategising and Firm-stakeholder Interaction in Open Source Software Communities. The ICEB & EBRF Conference Tampere, November 28- December 2, 2006.

13. Oksanen Ville, Helander Nina, Seppänen Marko, Puhakka Mikko & Laine Juha. 2007. Building SaaS Business on Top of Open Source – Economic and Legal Considerations. Berkeley-Tekes Innovation in Services Conference. April 27-28, 2007. Berkeley, USA.

14. Oksanen, V., Välimäki, M. & Laine, J. 2005. An Empirical Look at the Problems of Open Source Adoption in Finnish Municipalities. Seventh International Conference on Electronic Commerce, China.

15. Puhakka, M. & Jungman, H. 2006. Evaluation and Valuation of Open Source Software Companies: A Venture Capitalist' Perspective. In Seppä, M. Hannula, M. Järvelin, A-M, Kujala, J. Ruohonen, M. & Tiainen, T. (eds.). *Frontiers of e-Business Research 2005*. Tampere University of Technology & University of Tampere.

16. Puhakka M., Oksanen V. & Seppänen M. 2006. A study of the deployment of open source software – Finnish experiences from public and private sector. The ICEB & EBRF Conference Tampere. November 30th – December 2nd, 2006. CD-ROM.

17. Vainio, N., Vadén, T. & Oksanen, V. Sustainability of open collaboration communities: five aspects. FM10 Openness: Code, Science and Content. 2006.

18. Vainio, N., Oksanen, V. & Vadén, T. Company Participation in Open Source Software Communities: Measuring Sustainability. The ICEB & EBRF Conference Tampere, November 28-December 2, 2006.

Other Publications

19. Aaltonen, T. & Jokinen J. Demography of Linux Kernel Developers. Tampere University of Technology. Institute of Software Systems. Report 41.

20. Puhakka, M. The Future is Open – Are You and Your Company ready? forthcoming in 2006 Ariadne Capital Journal.

21. Puhakka, M. & Tapper, H. Open Source and Proprietary Software – The Finnish Experience. forthcoming in 2006. TIEKE.

22. Vainio Niklas, Oksanen Ville, Vaden Tere & Seppänen Marko. 2007. Elements of Open Source Community Sustainability. Poster at the OSS2007, 11-14 June. Limerick, Ireland.

23. Vainio, N. & Vadén, T. Valoa basaarista. Internetin vapaan ja avoimen koodin kollektiivinen kehitystyö. forthcoming 2006 in Parviainen, Jaana (ed.): Kollektiivinen asiantuntijuus. Tampere University Press.

Master theses

24. Ilkka Luoma. Forthcoming 2007. Master of Science Thesis. Tampere, Tampere University of Technology.

25. Myllärniemi, Jussi. 2007. Structures and operations of open source value networks. Master of Science Thesis. Tampere, Tampere University of Technology. 77 p.

26. Jarkko Laine. Forthcoming 2007. Master of Science Thesis. Tampere, Tampere University of Technology.

OSSI Reports

Helander, Nina. & Antikainen, Maria. (eds.) 2007. Essays on OSS Practices and Sustainability. e-Business Research Center Research Reports 36. Tampere University of Technology & University of Technology.

Helander, Nina. & Mäntymäki, Maria. (eds.) 2006. Empirical Insights on Open Source Software Business. e-Business Research Center Research Reports 34. Tampere University of Technology & University of Technology.

Helander, Nina. & Martin-Vahvanen, Hanna. (eds.) 2006. Multidisciplinary Views to Open Source Software Business. e-Business Research Center Research Reports 33. Tampere University of Technology & University of Technology.

Others

Helander, Nina & Ulkuniemi, Pauliina. (2006): Marketing Challenges in the Software Component Business. International Journal of Technology Marketing Vol. 1 No 4 pp. 375-392.

Mäkinen Saku & Seppänen Marko. 2007. Assessing business model concepts with taxonomical research criteria. *Management Research News*. Vol. 30 Iss. 10.

Seppänen Marko & Mäkinen Saku. 2007. Towards classification of resources for the business model concept. *International Journal of Management Concepts and Philosophy*. Vol. 2, Iss. 4.

Ahonen, M., Antikainen, M. and Mäkipää, M. 2006. What motivates customers to innovate for free? - Utilizing Web 2.0 Communities in mass customization and customer co-design. *Conference proceedings of Mass Customization and Personalization Forum 2006*. University of Tampere.

Ahonen, M., Antikainen, M. and Mäkipää, M. Supporting collective creativity within open innovation. EURAM 2007, Paris, May 16-19, 2007.

Mäkinen, S., Seppänen, M. & Nokelainen, T. 2005. Resources and Dynamic Capabilities in Business Model Concepts: A Review of the State-of-the-Art. The 25th SMS Annual International Conference. Orlando, Florida. USA. October 23rd-26th, 2005.

Mäkinen, S. & Seppänen, M. 2006. Strategic Management of Exploiting Technological Opportunities: Integrating Strategy to Operations with Business Model Concept. The 15th International Conference for Management of Technology Annual Conference 2006. Beijing, China. May 22nd – 26th, 2006.

Mäkipää, M., Ahonen, M. and Mäntymäki, M. 2006. Developmental steps from closed innovation to open innovation – Increasing customer involvement through mass customization and customer co-design. In *Proceedings of the 29th Information Systems Research Seminar in Scandinavia (IRIS 29)*. Helsingor, Denmark.

Seppänen Marko & Mäkinen Saku. 2007. Resource categorisation as a part of business model concept: an empirical assessment of appropriateness from business unit manager's perspective. The 14th International Product Development Conference. Porto, Portugal. June, 10-12, 2007.

Seppänen, M. 2006. A propositional inventory of human resources for the business model concept. The proceedings of the 3rd International Conference ENTIME 2006. Gdansk, Poland, September 22-23, 2006. 8p. CDROM.

Seppänen, M. & Mäkinen, S. 2006. Conceptual Schema of Resources for Business Models. The 3rd IEEE International Conference on Management of Innovation and Technology, Singapore, June 21-23, 2006.

Seppänen, M. & Mäkinen, S. 2005. Business model concepts: a review with case illustration. The Proceedings of the International Engineering Management Conference 2005. St. John's, Newfoundland, Canada. pp. 376-380.