Providing Commercial Open Source Software: Lessons Learned

Øyvind Hauge and Sven Ziemer
Norwegian University of Science and Technology
{oyvind.hauge|sven.ziemer}@idi.ntnu.no,
WWW home page: http://www.his.se

Abstract. Even though companies like Sun, IBM, MySQL and others have released several commercial Open Source Software (OSS) products, little evidence exist of how to successfully launch such products and establish a living community around them. This paper presents a case study from a small software company succeeding at establishing a business model and a vivid community around their own OSS products. Based on this case study, the paper presents lessons learned which could help other OSS providers.

1 Introduction

Open Source Software (OSS) development has become a serious source of revenue for the software industry [10, 13]. Large companies like Apple, IBM, Sun and others have released significant amounts of their software as OSS. Going open source can however be a significant change for a commercial organization [5]. Small and medium enterprises (SME) do not have the same resources as large companies to adapt to these changes. Yet, companies like JBOSS, MySQL and Qt Software have successfully established businesses around their own OSS products. Even though these OSS providers have been quite successful, the research literature contains only limited empirical evidence on the challenges and benefits which face a commercial OSS provider [27]. We define a commercial OSS as an OSS product being released by for-profit organizations like MySQL [7], Philips Healthcare [18], JBoss [25], and IBM, Apple and Sun [26]. While these large well known OSS providers have received some attention, small companies providing their own OSS products are overlooked. This is unfortunate since SMEs with less than 250 employees constitute almost 70% of the sector for computer and related activities in the European Union [9].

In this paper we present the story of a small Norwegian software company that has built their business around their OSS products. We analyze the case and compare the findings from this case with what has been reported in the literature. Based on this discussion we also present some lessons learned that may help other companies in their establishment of a viable business model around their own OSS products.

2 Related Works

Companies and organizations providing OSS have attracted some attention in the literature as for instance [1, 7, 16, 25, 26]. Nevertheless, research on commercial OSS providers is generally missing [27]. Here we discuss three important topics from this literature; business models, communities and software licenses.

2.1 Business Models and Related Issues

The ways companies approach OSS development are diverse [28] and several business models are described in the literature [10, 14, 17, 20]. Four such models are the value-adding service enabler, market creation, leveraging community development and leveraging the OSS brand [10]. Two of these models are particularly interesting for OSS providers (1) using an OSS product to create a market for other services and products and (2) getting contributions from the OSS community [26]. An OSS provider may also use OSS branding to promote its products. While service enabling is more appropriate for companies extending existing OSS communities rather than OSS providers seeking to create their own. Companies may also use OSS products to reach other strategic goals besides directly making money on them. The DICOM validation tool was released as OSS primarily to establish a de facto standard to save rather than to make money [18]. Moreover, Sun established the Java platform to limit Microsoft's control over industry standards [26].

Companies like MySQL and JBOSS do on the other hand build their business around their OSS products [7, 25]. Profiting from the OSS product and its community is for these companies particularly important. Thus creating or identifying a demand for one's products and services is one of the most important risks facing an OSS provider [25]. Roxen tried to make an OSS competitor to the Apache HTTP Server but was forced to change focus due to the strong position of Apache and a lack of demand for their own OSS solution [7]. To be able to create or identify a need for ones products, a commercial OSS provider must understand its customers and their domains. They should therefore hire developers with domain knowledge [27] and use their own software [15] to better understand its strengths and weaknesses.

Making adjustments to the business model and adapting to opportunities and challenges, is key for an OSS provider. When Firefox started to get popular, a wave of viruses and security issues came across the Internet and created a need for a new browser. Firefox was there to fill that need [1]. JBOSS has also been able to adapt to changing opportunities and customer needs [25]. First, the community requested training and documentation. Second, customers demanded advice on building Java applications on top of JBOSS. Third, customers wanted support. Fourth, customers all over the world needed local expertise. JBOSS has evolved its business model by providing training, documentation, consulting services, sup-port and finally an international partner program [25].

2.2 Community

Succeeding at attracting a community is one of the most difficult challenges related to releasing a commercial OSS product [7, 18]. Just releasing the source code is clearly not enough [1]. Considerable investment and several support functions may be needed to successfully release an OSS product [15, 27]. First, practical measures must be taken to prepare a product for release. The source code should be documented and written in a comprehensible manner so it can be understood by users and developers, and the product should be packaged and distributed in easy installable packages [2, 19].

Next, it is necessary to create a common infrastructure on which the company and the community can collaborate. The provider has to set up tools for easy communication and sharing of code, knowledge, experiences and problems [2]. In one project, the participants failed to agree on a configuration management strategy and a set of tools for version control [6]. This made development difficult and contributed to the failure of the project [6].

Another prerequisite for releasing an OSS product is a stable team of core developers which can secure the continuity of the project [15]. This core team should provide the necessary structure to keep the project moving forward [27]. The provider must have resources which can support the product's community including responding to questions and bug reports, fixing problems, take care of contributions and so on [2, 15]. Even though companies may release a product to get contributions from the community [10] most end up implementing almost all the code themselves [24]. A reason for this could be that it proves difficult to rely on the community performing mundane tasks like maintenance, support and so on [15]. Next, in many cases the company wants control over the product to be able to guarantee the quality of it to its customers. Furthermore, the company's employees work with the product the whole time and they are therefore the ones with the most extensive knowledge of it.

To run a community it must be included in the ways of the company, the community members must feel able to contribute to and influence the product, and the provider must respect the norms and values of an OSS community [7, 27]. The OSS norms and values must also be spread to the community, in particular other companies, as the idea of not sharing with other companies is still rooted in the culture of many companies [2].

To include the community, the provider must apply a governance model which is appropriate for the needs of all the stakeholders involved in the community [27]. Too much focus on only a group of stakeholders could be harmful in the long run [15]. Consequently, the provider must be open to new community members and make it as simple as possible to participate in the community [4, 15]. Open communication and transparency should help community members understanding the provider and ongoing activities. OSS projects should furthermore have well documented goals, roles and responsibilities [4]. When opening up the development around Mozilla, the development crew had to release more information and to use public information channels to include the community members [1]. In another project, the project team

wanted to deliver a mature product to the OSS community and decided to develop it internally before releasing a mature version [6]. This was a big mistake as communication with the community was very scarce during the development. External users were because of the lack of communication and a product, not particularly interested when the product was re-leased.

To encourage community contributions, the provider should also consider let-ting go of some control [1]. Too strict control over the product and the community may be counterproductive [18]. If necessary, payment or gifts could be considered to encourage certain behavior or to get contributions [7].

2.3 Software Licensing

Commercial OSS providers must apply a license which is fruitful for both the company and the community [7, 27]. The license must enable the company to make money on either the product or related services and it should enable the growth of a vivid community. A license which the users are unhappy with can severely limit the adoption of a product and it may provoke strong reactions from the community [12].

An OSS provider has a few choices when it comes to selecting a license, as he may develop new licenses or reuse existing ones. Creating new licenses is discouraged [11] since potential users will be unfamiliar with the new license, and since it would require significant resources to create a license of high quality. By reusing existing and well known licenses it is more likely that potential users are familiar with the license, that it is tested, and that it is of good quality.

When reusing existing licenses the OSS provider basically has three choices [8]. First, the OSS provider may use a license like GPL which requires all derivate products to be released under the same license. This may enable him to release the product under a proprietary license as well and thereby create an income from a dual licensing scheme [11]. However, a dual licensing scheme requires that the provider own intellectual property rights for the whole code base. Second, the OSS provider may select a license like MPL which requires direct changes to the original code base to be licensed with the same license, and thereby ensuring that bug fixes and similar changes done by others will be available. Third, the OSS provider may use a license like the new BSD license which sets no restrictions on the choice of license on derivate works, and thereby encourage adoption in any kinds of products.

3 Method

This paper reports on research performed in the COSI project. COSI stands for "Co-development using inner & Open source in Software Intensive products" and is a European industrial research and development project. The project ran for three years, from November 2005 until October 2008 and was organized as a con-sortium of 13 industrial and academic partners from five countries. The project’s goal was to increase awareness of industrial usage of distributed collaborative software and OSS.

The research design of the COSI project consisted of five phases, including two case executions, where the companies were working on selected issues identified by the project's plan. During the case executions the companies documented their practices, identified problematic issues and improved these practices.
The authors worked with the five Norwegian companies in the project, supporting and guiding their activities in the project. In addition, we collected data relevant for OSS research. In the case of eZ, the activities were focused on under-standing and improving the community management practice, and both case executions addressed this issue.

This research has applied two methods for data collection in this approach: the qualitative research interview and post-mortem analysis (PMA). In addition, we had access to the project deliverables from eZ and had also several informal meetings with the company at COSI workshop meetings, community conferences and other occasions.

Eleven interviews have been conducted with four persons from the development group from eZ at several occasions, distributed over the three years the project lasted. The interviews have been unstructured [21] and have been focused on both on the current community management practice and the history of eZ's main product eZ Publish (hereafter Publish). Notes were taken from all interviews and sent to the interviewees for review.

The authors organized two PMA [3] sessions with most of the developers in the development team. Both sessions focused on how the community management process could be changed in order to increase the number of community contributions to Publish. During these sessions we described the current community management practice and identified both positive and negative issues with this practice. In addition root-cause analyses for some of the negative issues were conducted.

This paper presents the story of a SME that has successfully developed an OSS product and attracted a large community that contributes substantially to the ongoing development of the product. The authors had access to eZ for more than three years. During this time an understanding of how eZ was able to make these achievements was built up based on the conversations with the employees and the authors' reflection. As mentioned above there is little literature on how SMEs develop OSS products, what business models they choose and how they create and take advantage of a community to develop an OSS products. This paper shares lessons learned from such a company and contributes thus to a broader under-standing of how SMEs can release OSS products and used the products to attract a community of users and developers.

In analyzing the data and identifying potential lessons learned we found that there are two ways of understanding of eZ's achievements. The first way of under-standing is the one of the interviewees, who presented the development of Publish as a series of strategically planned activities. The second way of understanding is from the authors, who see the development of Publish not as a strategic planned activity but rather driven by the skill to identify new opportunities and to make rapid

decisions to realize the opportunities. It is the authors' view that both understandings are equally valuable and needed to attract and take advantage of a community.

4 The eZ Systems Case

eZ Systems is a Norwegian software provider founded in 1999. Today they have around 60 employees spread over offices in Norway, Denmark, Germany, France and North America. eZ has almost since its origin focused on providing a PHP based OSS Content Management System (CMS), eZ Publish. The company has a large customer base from all over the world and the CMS has been downloaded more than 2.5 million times from their web site, as of February 2009.

4.1 The Early Days 1999-2001

In the beginning, eZ focused on developing applications for stock brokers but delivered at the same time consultant services to local businesses. These services included network and systems administration, and application and web development. The increasing popularity of the Internet gave them several customers who wanted web sites. Many of these sites contained similar functionality and eZ soon started reusing code from one site to another. This reusable code was quickly bundled into two packages, Publish (article management) and Trade (shop management) and released under the GPL, see Figure 1. The employees' support for the OSS ideology made releasing the packages as OSS, natural.

The company continues developing stock market applications. Meanwhile, the CMS attracts attention in the OSS community and requests for consulting services related to Publish are coming in. In parallel, they start selling the OSS philosophy to local businesses. The philosophy is simple, if eZ disappears or if the customer is unhappy with eZ's work, he has access to the source code and he may hire some-one else. Publish is an attractive product and as a consequence of growing interest from both customers and the OSS community, Publish gradually requires more and more attention. This growing interest forces them to focus on either the stock market applications or Publish. Even though it is a bold move including significant risks, the final decision is to discontinue the stock market application and focus 100% on Publish. The employees have a strong desire for OSS, they really want to create a viable business model based on OSS, and releasing an OSS product sounds fun.

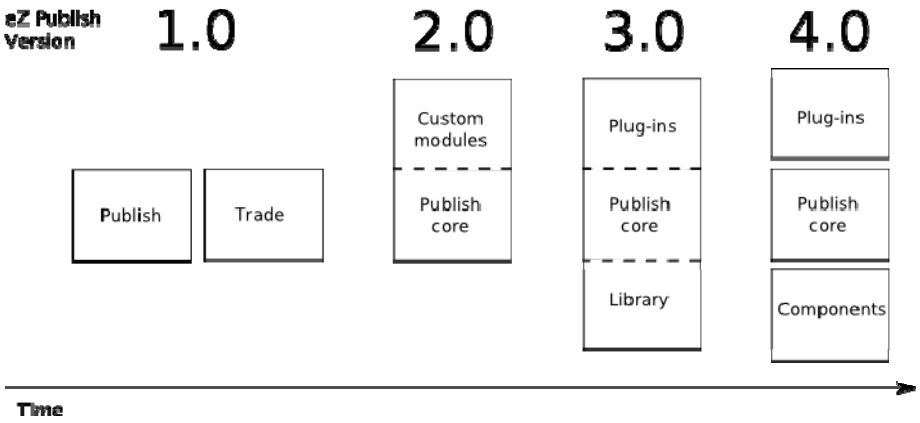

Fig. 1. The development of the Publish architecture

4.2 The Middle Ages 2001-2005

After deciding to focus on the development of the CMS, eZ starts developing Publish 2.0. This version is module based with the intention of enabling custom modules extending the core functionality, see Figure 1. However, the possibility to extend existing modules without changing the kernel is very limited, if existing. Even though there are some problems with the modular architecture, the system provides interesting functionality, and it therefore attracts a rather large community of OSS users.

The development of the third version starts in 2003 and the PHP 4 based 3.0 version is released in March the next year. The focus of this version is increasing the modularity of Publish, allowing Plug-ins and simplifying the configuration of the system. A simple two layer architecture consisting of a library and the application itself is attempted in addition to the plug-ins, see Figure 1. However, the two layers are soon too dependent of each other, making it eventually impossible to use the library without installing the application. Even though eZ is unable to keep the two layers separated the plug-in architecture is a success in the sense that it enables the users of Publish to extend it with their own functionality.

4.3 Components and Publish 4.0 2005-Today

Due to dependency problems in Publish it is decided to make a new independent library, giving birth to eZ Components (hereafter Components), see Figure 1. The library is built separately from the CMS and the development process is opened up to the community. The idea is to create a library which could be used for a wide variety of PHP applications. The library should also be included into Publish when it

reached a mature state. This is done iteratively to straighten out eventual problems one at a time. The Library is furthermore a way of refactoring the code in Publish, gradually introducing PHP 5 to the CMS and ensuring support for Windows, Unix and Linux. Late 2007, the forth major version of Publish is released. Through refactoring of Publish and by incorporating Components into the CRM, it gains PHP 5 support. Components furthermore enables those making plug-ins for Publish to make use of the functionality it provides and thereby achieving synergies between the two communities. The division of the system into independent parts enables the growth of three communities around Components, Publish and the plug-ins, see Figure 2.

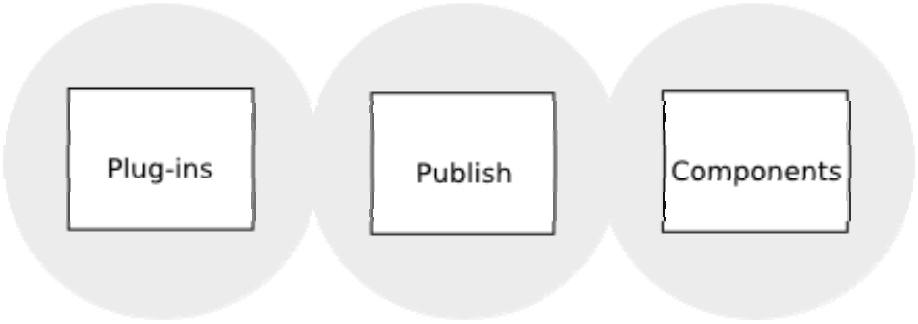

Fig. 2. The parts of eZ Publish and their surrounding communities.

5 Analysis of the eZ Case and Comparison with Findings from the Literature

In the previous section we gave an historical overview of how a small Norwegian software company has successfully launched an OSS product and attracted a large ecosystem of users and developers. This ecosystem can, as illustrated in Figure 2, be divided into three communities. In this section we will review the case, using the challenges identified in the literature.

5.1 Business Model and the Benefits of Communities

Having a large number of potential customers in the community around Publish creates a greater need for services like support, quality assurance, training, installation, and hosting. Furthermore, it makes selling these services easier and reduces the need for marketing. Users are made aware of Publish through the Internet and services are often sold through bottom-up adoption of the product.

Advantages like reduced marketing efforts and shorter sales cycles are also observed else-where [19, 25].

The Plug-ins community has developed a large number of plug-ins which ex-tend the functionality of Publish. These plug-ins increase the whole value of the product, enable community members to solve their specific problems, and help eZ to understand these problems. Furthermore, one might see the activity in the Plug-ins community as a way of outsourcing the development and maintenance of these plug-ins, and thereby reducing eZ's development efforts. The community members' investments in developing these plug-ins build a stronger connection between them and Publish and thereby increase their loyalty to it.

The Components community contributes code to a library eZ would have needed to develop regardless of these contributions. Next, the future of PHP is essential to eZ's products and Components, particularly if it becomes widely adopted, is a tool eZ can use to keep up with and influence the development of PHP. Adoption of the library will also contribute positively to increasing eZ's reputation, particularly in the OSS community.

Using the categorization of business models in [10] we see that the communities around eZ support different strategic goals. Publish is creating a market for the supplementary services eZ and their partners provide. More, through the two other communities, eZ gets contributions from the OSS community. OSS products can as we see be used to reach other strategic goals than directly increasing the income of a company [18, 27]. Components, the plug-ins and their communities illustrate this as they contribute to reducing eZ's development costs, increasing the value of Publish, and to monitoring and influencing the future of PHP. eZ are in other words using different strategies for each of the communities to support their over all business strategy.

eZ is furthermore able to construct a good understanding of the needs of their users through feedback, requirements and interaction with all three communities. Community developed plug-ins, recruitment of developers from the community and the use of their own product give eZ better understanding of the domain and thereby reduce their expenses on market research.

The business strategy of eZ has evolved from application development targeting a specific domain to providing services and support to the ecosystem around an OSS product. An evolution of the business model can also be seen in the JBOSS case [25]. Income from services and support are more predictable and consistent than from licenses and consulting, and less sensitive to economic turn-around [25]. This is being particularly true when having a large install base. It is therefore natural to evolve the business model as the customer base grows.

5.2 Community

Infrastructure: eZ has been investing in a common infrastructure for the three communities. For the Plug-in community, eZ is hosting a portal for plug-ins, as well as organizing developer days at their annual Publish event. The infrastructure for

Publish consists of forums, mailing lists, issue trackers, documentation and source code. For the Components community mailing lists and an open issue tracker are provided.

Providing this infrastructure is a rather small investment, even for a small company. In addition, eZ did not set up their community infrastructure before the product was released but did so over time, driven by the activity level and demand of the communities. The cost of establishing the infrastructure has thus been spread out over several years. This contrasts the findings of [15, 27], that both mention that considerable investment is needed to release an OSS product and to set up support functions. One possible explanation is that eZ never planned from the start to provide an OSS product.

Attracting and governing a community: Attracting and governing a community is one of the most challenging aspects of releasing an OSS product [7, 18]. Today eZ has an ecosystem that consists of three communities, serving its two products. Together this ecosystem attracts users, volunteers and customers to use the products and to be part of the communities. eZ is attracting the communities by providing two interesting products that are downloaded and used by a large user base. eZ is further attracting member to their communities by accepting and hosting plug-ins to their Publish product, and by accepting contributions from both the Publish and Components community. eZ also communicates a positive attitude towards open source to the outside world and uses the open source label to differentiate itself from non-open source competitors.

Attracting a community starts with releasing an attractive product, that is of interest to a potential large user base. The most active community in eZ ecosystem is the Plug-in community. It started when users started developing their own functionality by using the plug-in mechanism in the architecture of Publish. These developers wanted to share their plug-ins with other Publish users, and reflected thus the same attitude to open source that made eZ release Publish as an open source product in the first place. The plug-in community is attractive to its members even when the members are not included in the way of the company. The inclusion in the way of the company is suggested to be a necessity to attract a community [7, 27]. eZ is including the members of the Publish and Component communities in varying degrees, but in none of the communities are the members fully included in the way of the company. The community members' motivation to contribute is thus not the inclusion in the way of a company but rather implementing functionality they are interested in themselves. The argument made here is not that it is not important to include community members into the ways of a company, but that the attraction of a community starts with a product that is appealing to a large number of users.

eZ is as of now not satisfied with the activity level in the Publish community and would like to increase it. This deals with how to govern a community, and with how to balance conflicting interests between the community and eZ customers. Since Publish is the strategic core product for eZ, control with the product and its future development is needed for strategic reasons. Exercising too much control, however, may result in that the community looses its attractiveness with its members [18].

5.3 Software Licensing

Publish and Components address different strategic goals. To avoid licensing problems, to attract a community and to reflect these strategic goals, eZ selected two different, but well established OSS licenses. The GNU Public License (GPL) allows the community to use Publish without paying any license fees. At the same time it gives eZ control over how Publish is used. GPL requires code sharing and prevents the use of the source code in proprietary products. Moreover, GPL enables eZ to dual license Publish and thereby getting some income from the license sales. eZ provides proprietary licenses for companies which (1) include Publish in their proprietary products, (2) build proprietary extensions on top of Publish and (3) use Publish as any other proprietary software. This last license is particularly useful for companies which not yet have legally approved the use of OSS licenses in their organization. However, to lower the threshold for adoption of Components, eZ released it under the New Berkeley Software Distribution (BSD) license which gives adopters quite unlimited freedoms.

6 Lessons learned

With eight years of experience, the eZ case identifies some lessons learned about how to release an open source product.

Allow your business model to evolve: Providing an OSS product is not a trivial task, and the experience from the eZ case shows that providing an OSS product may take unexpected turns. Even though the use of OSS in the software industry is growing, OSS business models have yet to stabilize themselves. It is thus important to plan for a business model and to allow it to evolve with the opportunities and challenges presented by the product and its community. Core team needs experience from other OSS projects and communities Setting up an OSS community requires knowledge about how open source communities function. Having developers with experience from other open source communities is beneficial since they have fit hand experience with OSS values and practices. The Components com-munity is a good example that this is helping to create an active community.

Balance control and bureaucracy related to community contributions: Lack of control over community contributions directly to your product can reduce the quality of it and potentially introduce illegitimate source code into the product. Too strict control on the other hand may discourage contributions and community participation. It is therefore important to clearly specify where you are going with your product and what kind of contributions you want, and to make contributions and wanted behavior visible to other community members.

Be part of your own community: In order to sustain a community of volunteers a community needs to be active and including. This can be achieved when the core development team is part of the community, and uses a common infrastructure to share information and to co-ordinate all activities. This creates the transparency that

a community is expecting. The opposite of such a transparent community would be a community where the core team uses a parallel infrastructure to communicate and co-ordinate their activities internally.

Apply well known licenses which suit both you and your users: Unnecessary strict licenses may limit the adoption of a product. Both OSS users and paying customers will most likely go elsewhere if their needs are not met by the software's license. To avoid intimidating the users, simple, well known licensing models should be chosen. Explain the OSS licenses, its permissions and restrictions. Launching a product as OSS could include a constant fear of license infringement. When the source code is available it is technically quite simple to misuse the source code. However, this has not been a problem for eZ and the very few incidents which have occurred have easily been solved.

7 Discussion and conclusions

Finally, some issues will be pointed out. First, investing in an infrastructure is not reserved only to open source providers. While this investment has been seen as something that is an extra investment for companies providing OSS, providers of commercial products need an infrastructure as well to stay in touch with their customers, and receive error reports and other feedback.

eZ Systems have established an ecosystem with three communities that are based on different business models and give different benefits in return. This strategy to create more than one community with an OSS product seems to enable eZ to take advantage of several of the benefits that are associated with having a community of users. This division helps attracting and directing contributions to two areas where it is more convenient to receive them while controlling the core product. At the same time as eZ wants to attract more contributions to Publish (the core), there is also a need to keep certain control with this product for commercial reasons. Resolving conflicts between community interests and commercial interests is a delicate balance.

This paper has presented the history of the two open source products provided by eZ and the three communities that constitute the ecosystem around these prod-ucts. Based on eZ's experience, we have identified some lessons learned which could help other OSS providers. There is no single answer on how to succeed as an OSS provider. In case presented in this paper, however, there are some factors that contributed to the success of the provided OSS. This includes the evolvement of the business model, having an attractive product and adapting to community needs and opportunities.

Acknowledgments

The research has been conducted within the ITEA COSI project and is supported by the Research Council of Norway. We are grateful for the support from eZ Systems

and to our colleagues in the COSI project, in particular Vidar Langseid, Thomas Østerlie, and Carl-Fredrik Sørensen.

References

1. Mitchell Baker. The Mozilla Project: Past and Future. In Chris DiBona, Danse Cooper, and Mark Stone, editors, open sources 2.0, pages 3-20. O'Reilly Media Inc, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2006.
2. Massimo Banzi, Guido Bruno, and Giovanni Caire. To What Extent Does It Pay to Approach Open Source Software for a Big Telco Player?. In Russo et al. [22],pages 307-315.
3. Andreas Birk, Torgeir Dingsøyr, and Tor Stålhane. Postmortem: Never Leave a Project without It. IEEE Software, 19(3):43-45, 2002.
4. Wolf-Gideon Bleek and Matthias Finck. Ensuring Transparency - Migrating aClosed Software Development to an Open Source Software Project. In IRIS'28 Proceedings of the 28th Information Systems Research Seminar in Scandinavia, 2005. 6-9 August.
5. Wolf-Gideon Bleek, Matthias Finck, and Bernd Pape. Towards an Open Source Development Process ? Evaluating the Migration to an Open Source Project by Means of the Capability Maturity Model. In Scotto and Succi [23], pages 37-43.
6. Cornelia Boldyreff, David Nutter, and Stephen Rank. Communication and Conflict Issues in Collaborative Software Research Projects. In Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim R. Lakhani, editors, Collaboration, Conflict and Control Proceedings of the 4th Workshop on Open Source Software Engineering, pages 14-17, 2004.
7. Linus Dahlander and Mats G. Magnusson. Relationships between Open Source Software Companies and Communities: Observations from Nordic Firms. Research Policy, 34(4):481-493, 2005.
8. Paul B. de Laat. Copyright or copyleft?: An analysis of property regimes for software development. Research Policy, 34(10):1511-1532, 2005.
9. Eurostat: Number of persons employed by enterprise size-class in the EU-27, 2009. Online: http://ec.europa.eu/eurostat/, accessed 2009-02-12.
10. Brian Fitzgerald. The Transformation of Open Source Software. MIS Quarterly, 30(3):587-598, 2006.
11. Karl Fogel. Producing Open Source Software: How to Run a Successful Free Software Project. O'Reilly, 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2005.
12. Jim Hamerly, Tom Paquin, and Susan Walton. Freeing the Source: The Story of Mozilla. In Chris DiBona, Sam Ockman, and Mark Stone, editors, Open Sources:Voices from the Open Source Revolution, pages 197-206. O'Reilly, 1999.
13. Øyvind Hauge, Carl-Fredrik Sørensen, and Reidar Conradi. Adoption of Open Source in the Software Industry. In Russo et al. [22], pages 211-222.
14. Richard E. Hawkins. The economics of open source software for a competitive firm. NETNOMICS, 6(2):103-117, August 2004.
15. Juha Järvensivu and Tommi Mikkonen. Forging A Community Not: Experiences On Establishing An Open Source Project. In Russo et al. [22], pages 15-27.
16. Chris Jensen and Walt Scacchi. Collaboration, Leadership, Control, and Conflict Negotiation and the Netbeans.org Open Source Software Development Community. In HICSS'2005 Proceedings of the 38th Annual Hawaii International Conference on System Sciences, page 196b. IEEE Computer Society, 2005.

17. Sandeep Krishnamurthy. An Analysis of Open Source Business Models. In Joseph Feller, Brian Fitzgerald, Karim R. Lakhani, and Scott A. Hissam, editors, Perspectives on Free and Open Source Software, pages 279-296. The MIT Press, Cambridge, Massachusetts, 2005.
18. Juho Lindman and Topi Uitto. Case study of company's relationship with open source community in open source software development. In IRIS'31 Proceedings of the 31st Information Systems Research Seminar in Scandinavia, pages 1-22, 2008.
19. Alberto Onetti and Fabrizio Capobianco. Open Source and Business Model Innovation. The Funambol Case. In Scotto and Succi [23], pages 224-227.
20. Eric Raymond. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly, Sebastapol, CA, 2001.
21. Colin Robson. Real World Research. Blackwell Publishing, 2nd edition, 2002.
22. Barbara Russo, Ernesto Damiani, Scott A. Hissam, Björn Lundell, and Giancarlo Succi, editors. Open Source Development Communities and Quality IFIP Working Group 2.13 on Open Source Software September 7-10, 2008, Milano, Italy, volume 275 of IFIP International Federation for Information Processing. Springer, 2008.
23. Marco Scotto and Giancarlo Succi, editors. OSS'2005 Proceedings of The First International Conference on Open Source Systems, 2005.
24. Anthony I.Wasserman and Eugenio Capra. Evaluating Software Engineering Processes in Commercial and Community Open Source Projects. In Andrea Capiluppi and Gregorio Robles, editors, FLOSS '07 First International Workshop on Emerging Trends in FLOSS Research and Development, page 1, Washington, DC, USA, May 2007. IEEE Computer Society.
25. Richard T. Watson, Donald Wynn, and Marie-Claude Boudreau. JBoss: The Evolution of Professional Open Source Software. MIS Quarterly Executive, 4(3):329-341, September 2005.
26. Joel West. How open is open enough?: Melding proprietary and open source platform strategies. Research Policy, 32(7):1259 - 1285, 2003.
27. Joel West and Siobhán O'Mahony. Contrasting Community Building in Sponsored and Community Founded Open Source Projects. In HICSS'2005 Proceedings of the 38th Annual Hawaii International Conference on System Sciences, page 196c. IEEE Computer Society, 2005.
28. Sven Ziemer, Øyvind Hauge, Thomas Østerlie, and Juho Lindman. Understanding Open Source in an Industrial Context. In SITIS'2008 Proceedings of the 4th IEEE International Conference on Signal-Image Technology & Internet-Based Systems, pages 539-546. IEEE Computer Society, 2008.