Tampereen teknillinen yliopisto. Ohjelmistotekniikan laitos. Raportti 19
Tampere University of Technology. Department of Software Systems. Report 19

Imed Hammouda & Björn Lundell (Eds.)

Proceedings of SOS 2011: Towards Sustainable Open Source

TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Tampereen teknillinen yliopisto. Ohjelmistotekniikan laitos. Raportti 19
Tampere University of Technology. Department of Software Systems. Report 19

Imed Hammouda & Björn Lundell (Eds.)

Proceedings of SOS 2011: Towards Sustainable Open Source

Tampere University of Technology. Department of Software Systems
Tampere 2011

ISBN 978-952-15-2718-0
ISSN 1797-836X

Preface

This is a collection of the papers presented at the Towards Sustainable Open Source Workshop (SOS 2011). The workshop was co-located with OSS 2011 and was held in Salvador, BA, Brazil on October 08, 2011. The goal of the workshop is to build a community of researchers and practitioners to share experiences and discuss challenges involved in building and maintaining sustainable open source communities.

The program contained 5 technical presentations discussing topics related to forking, organizational issues, legality matters, digital archiving, and open innovation.

We would like to thank the members of the organizing committee and the program committee for their effort. We also appreciate the contribution of the authors of papers submitted.

We hope that you enjoy the workshop!

Tampere, December 2011

Imed Hammouda & Björn Lundell

Organizing Committee

Imed Hammouda, Tampere University of Technology, Finland
Björn Lundell, University of Skövde, Sweden

Program Committee

Andrea Capiluppi, University of East London, UK
Barbara Russo, Free University of Bozen - Bolzano, Italy
Björn Lundell, University of Skövde, Sweden
Daniela Cruzes, Norwegian University of Science and Technology, Norway
Imed Hammouda, Tampere University of Technology, Finland
Jonas Gamalielsson, University of Skövde, Sweden
Mohamed Amine Chatti, RWTH Aachen University, Germany
Scott Hissam, Software Engineering Institute, Carnegie Mellon, USA
Sulayman K Sowe, UNU-IAS, Japan
Tommi Mikkonen, Tampere University of Technology, Finland
Walt Scacchi, University of California, Irvine, USA

Contents

Forking: the Invisible Hand of Sustainability in Open Source Software

Linus Nyman[1], Tommi Mikkonen[2], Juho Lindman[1], and Martin Fougère[1]

1 Hanken School of Economics, Helsinki, Finland
firstname.lastname@hanken.fi

2 Tampere University of Technology, Tampere, Finland
tommi.mikkonen@tut.fi

Abstract. The ability to create and maintain high-quality software artifacts that preserve their usability over time is one of the most essential characteristics of the software business. In such a setting, open source software offers excellent examples of sustainability. In particular, safeguarding mechanisms against planned obsolescence by any single actor are built into the very definition of open source development. The most powerful of these safeguarding mechanisms is the ability to fork the project as a whole. In this position paper, we argue that the possibility to fork any open source program serves as the invisible hand of sustainability, ensuring that the code can always remain open and that the code that best fulfills the needs of the community will live on.

1 Introduction

Sustainability is a concept which is often automatically associated with open source software. Indeed, access to the source code enables developers to build solutions that are better protected from the actions of any single developer, organization, or company associated with the software. The openness of the source code also guarantees that decisions concerning the software artifact enjoy a measure of transparency.

In this position paper we address the role of code forking – a situation in which several versions of a piece of software originating from a single, shared code base are developed separately – in ensuring the long-term sustainability of a software system. Furthermore, we advocate the freedom that developers have to create novel features that may well go beyond what the developers who began the project originally expected or planned. This freedom will also nurture open source projects through difficult times and extreme events that could otherwise prove lethal, such as hostile commercial acquisitions which may cause changes in the practices of the project.

The rest of the paper is structured as follows. In Section 2, we briefly address the sustainability of digital objects, focusing on open source and planned obsolescence,

which is generally associated with almost all of the systems in common use. In Section 3, we discuss code forking, which can serve as an element that supports long-term sustainability. In Section 4, we offer our view of the effect code forking has on sustainability. Finally, in Section 5, we draw some of the main conclusions.

2 Sustainability and planned obsolescence

The sustainability of a product can be interpreted in many ways. From the viewpoint of the consumer, there are at least two central elements: quality and staying power. In other words, we often seek a high-quality product which remains usable for as long as possible.

This view of sustainability contrasts with what is known as "planned obsolescence", a term popularized in the 1950s by American industrial designer Brooks Stevens [1]. Stevens defined planned obsolescence as the act of instilling in the buyer "the desire to own something a little newer, a little better, a little sooner than is necessary" [2]. From the fashion industry, where last year's models are designed to look out-of-date by the time this year's models come around, to the software industry, where the norm is for software to be compatible with older models, but not with newer ones, planned obsolescence has become an inescapable part of the consumer's everyday life.

Of course, digital artifacts differ substantially from the end products of 1950s industrial design, or even those of today. The main differences are related to their characteristics as editable, interactive, reprogrammable, distributed, and open [4]. These characteristics dictate that for example, software as an artifact is prone to being changed, repaired and updated rather than remaining fixed from the early stages of the design process. The software marketplace has transferred planned obsolescence to the digital realm by creating ways to benefit from these artifact characteristics. The revenue models of companies that operate in the software marketplace thus welcome versioning, lock-ins, systems competition, and network effects [6].

Open source software offers an alternative to some of the pitfalls of planned obsolescence. Rather than needing to buy something "a little newer, a little better", the open source community can simply make the existing product a little – or a lot – newer and better. In open source, anything, once invented, once written, need never be rewritten. On the other hand, the software product is never ready, but can become stable and mature enough for the developer community. If the community interest is there, the software can always be improved.

The right to improve a program, the right to make it portable to newer as well as older programs and versions, and the right to combine many programs into an even better whole are all rights built into the very definition of open source. The net result is that, in open source systems, any program which has the support of the open source community will enjoy assured relevance rather than planned obsolescence.

In fact, planned obsolescence in open source is impossible to implement due to a practice which is at once both the sustainer and potential destroyer of open source programs: the code fork.

3 Code forking

A popular metaphor in economics is Adam Smith's "invisible hand" which guides the marketplace. We claim that open source software has its own invisible hand: the fork. In fact, even the *possibility* of a fork – something which is guaranteed by all open source licenses – usually suffices. Actual forks are rare, but it is enough that they *could* happen, should the conditions in which the project is being developed change radically. In recent years, examples of using a fork for the sustainability of a community include high-profile cases such as the forking of OpenOffice (http://www.openoffice.org/) into LibreOffice (http://www.libreoffice.org/) and the creation of various projects from the code base of MySQL (http://www.mysql.com/).

A broad definition of a code fork is when the code from an existing program serves as the basis for a new version. This can be the result of a split in the developer community regarding the software artifact, its development practice, or the direction of the development, and is then usually followed by a split in the user community. Code forking in open source software is paradoxical in nature; it is simultaneously both one of the greatest threats an individual project faces, as well as the ultimate sustainer: insurance that, as long as users find a program useful, the program will continue to exist.

The threat to the program comes mainly in the form of the (potential) dilution of both users and developers. As Fogel [3] has noted, it is not the existence of a fork that hurts a project, but rather the loss of developers and users. The benefits of a fork come in ensuring that the program can continue to exist regardless of external circumstances. If, for instance, the developers of a program under a permissive license decide to relicense it under either a proprietary or otherwise less favorable license, the community can fork a new version and continue development. In the early days of open source, forking enabled the community to choose which version of UNIX to adopt. Forks can also serve as an escape hatch for projects and developers who find themselves cornered or unable to continue on a planned course.

In the case of a program remaining under an open source license, but where the people or company shepherding the code make decisions which run counter to the interests of the larger community and developers, code forking ensures the continued development of the code, as the community and developers can fork a new version on which to continue working. Even situations in which different versions of a program fork live on can benefit one another, as one community can incorporate into their program anything developed by the other community.

4 Code forking and sustainability

With open source, one can always fork a project; the code is available for download and the open source licensing terms impose no conditions which would in any way require developers to adhere to the original development line. In successful projects, however, a balance of power seems to exist where developers are happy enough to follow the project leader as long as the project leader listens to developers' views enough to keep them onboard. This balance creates continuity for long-term cooperation.

The mere possibility of forking has a huge impact on how open source programs are governed and developed [3], and provides the community with the tools it needs to handle situations in which a program could become obsolete. This can happen for numerous reasons, including the creation of a new version of the system, a change in licensing, porting to a new hardware environment, a change in program focus, and so forth.

For an open source project to remain sustainable, it must evolve with its users. Code forking and, indeed, the mere possibility of forking, is one of the key factors that ensures that open source will continue to evolve and thus remain sustainable. Open source programs can also cease to develop; some programs and pieces of code live on while others die out. Forking, as well as the effect of the possibility of forking, ensures that the selection lies in the hands of the community itself, which is perhaps the greatest guarantee of sustainability one could possibly ask for. At its best, open source software, guided by the invisible hand of forking, may well render obsolescence itself obsolete.

5 Conclusions

In this position paper, we argue that forking has the capability of serving as an invisible hand of sustainability that helps open source projects to survive extreme events such as commercial acquisitions, which may dramatically affect licensing and community support practices. While forking can occur for numerous reasons, some of which are less dramatic than others (see [5] for a survey of SourceForge projects), the mere possibility of forking is a powerful incentive for ensuring continuity.

To summarize, we claim this invisible hand is an essential element for the long-term viability of a project's development and thus the sustainability of the resulting open source software artifacts, and that without the opportunity to fork, many events now often considered mere annoyances could lead to the termination of a project.

References

[1] Planned obsolescence, The Economist, 23 March 2009. Available at: http://www.economist.com/node/13354332, accessed 14 September 2011.
[2] Brooks Stevens biography, available at: http://www.brooksstevenshistory.com/brooks_bio.pdf, accessed 14 September 2011
[3] Fogel (2006) Producing Open Source Software. O'Reilly, Sebastopol, CA.
[4] Kallinikos, J., Aaltonen, A., and Attila. M. (2010). A theory of digital objects. First Monday, Volume 15, Number 6-7 June 2010.
[5] Nyman, L. and Mikkonen, T. (2011) To Fork or Not to Fork: Fork: Motivations in SourceForge Projects. Proceedings of the 7th International Conference on Open Source Systems (OSS 2011), 259-268, Springer.
[6] Shapiro, C., and Varian, H. (1998). Information Rules: A Strategic Guide to the Network Economy. Boston, MA: Harvard Business School Press.

The Inextricable Role of Organizational Sponsorship for Open Source Sustainability

Carlos D. Santos Jr.[1], George Kuk[1], Fabio Kon[2] and Rafael Suguiura[2]
1 Horizon Institute, University of Nottingham, United Kingdom
{carlos.denner, george.kuk}@nottingham.ac.uk
WWW home page: www.horizon.ac.uk
2 FLOSS Competence Center (CCSL), University of Sao Paulo, Brazil
fabio.kon@ime.usp.br, suguiura@usp.br
WWW home page: ccsl.ime.usp.br

Abstract. Is the Bazaar a step to the Cathedral? This essay points out that organizational sponsorship appears to be inseparable of commercial grade, long-lasting open source software; and discusses the implications of that for organizational theory (rise of firms) and open source practice (IT governance).

Keywords: Open source, free software, sponsorship, sustainability, governance.

1 Introduction

We grew used to the metaphorical image that a group of grassroots volunteers were in charge of developing free and open source software (OSS). However, this metaphor had to be adapted to accommodate the increasing contributions from organizations to the production of OSS. The presence of organizational sponsorship had profound effects on how we perceived and should study the structure of these projects, the coordination mechanisms in place, and the motivations of contributors. The readily available findings of research on volunteerism and the Bazaar illustration did not quite fit the OSS phenomena as we initially thought. A more specific and thorough analysis of these communities was pressing to address the literature bias towards the study of volunteers' motivations as opposed to organizations' (Santos Jr., 2008). The first scholars to notice this gap between what the literature stated and what was being observed in OSS communities called for new research and proposed a label change, from OSS 1.0 to OSS 2.0, in an attempt to explicitly state that organizations were heavily involved in OSS, being thus partly responsible for their readiness for professional adoption (Fitzgerald, 2006; Watson et al., 2008).

A few years passed and now we have incorporated in the literature an updated image that there is a mix of volunteers and organizations in charge of OSS production, particularly in those that manage to build a productive ecosystem. Frequently, we see industry indicators of trustworthiness in OSS to take into account the presence and identity of organizational sponsors (e.g., see the Qualipso[1] process), and papers have been written to support organizations involved in OSS to effectively manage their relationships with the community of volunteers and industry partners towards sustainability (Agerfalk & Fitzgerald, 2008; West & O'Mahony, 2008). The

[1] http://www.qualipso.org/node/558

current state of the literature reflects the facts that: OSS has been acquired by organizations; OSS projects (OSP) have themselves become legal bodies capable of having employees, attracting partners and funding, and of managing a portfolio of projects; and that OSPs are commonly born out of software developed by organizations engaged in the currently popular strategy of opensourcing. Altogether, these observations suggest that successful OSPs are (and should strive to be) collaborative efforts between organizations and a community of volunteers, having the shape of a boundary spanning unit and being thus secondary to the higher organizational missions of the entities involved in the development.

The idea of becoming a formal organization to seek and accommodate sponsors in the production of OSS is nowadays so obvious that there is reason to question the sustainability of this software development model without it. Empirical evidence suggests that all OSS that are candidates for adoption at a professional level enjoy organizational support of some kind. Key OSS such as Linux, Apache and Android represent alliances of major industry players involving Google, Facebook, IBM, Yahoo! and Microsoft, to name a few. Moreover, organizations created to support potential adopters of OSS in the processes of selection and implementation tend to stamp only those that have secured sponsorship and met the legal and managerial conditions to sustain it. As a consequence, organizational involvement in OSS has superseded the role of volunteers, who now have to strive for sponsorship and collaborate with corporations if their projects are to build a market-wide reputation. Therefore, it seems timely to ask: Is organizational sponsorship a required feature for open source sustainability? Does sponsorship-seeking lead to the design of a formal organization (e.g., foundation)? Also, wasn't the first image we had of OSS as a volunteer-based effort trustworthy? What are the limits of a community-exclusive, Bazaar-like effort to develop OSS? Is there an inherent need to institutionalize OSS, moving away from a market-type of governance structure to sustain its development?

The goal of this paper is to think-provoke and instigate scholars to pursue a greater understanding of what a dispersed community of volunteers can produce by means of self-organization without creating or relying on formal organizations to sustain work. Yet, the limits of the Bazaar are unknown and our acceptance of the role of organizations in OSS production has gone unquestionable and assumed unavoidable to produce professional software of high quality.

Next, we discuss our current understanding of what constitutes a contribution and what the motivations to contribute are in the context of OSS. That discussion provides the grounds we needed to foresee a few implications for organizational theory and open source practice, exposing when and why a market-type of governance collapses, giving rise to a firm with characteristics of ephemeral alliances.

2 Contributions, Motivations and Sponsorship: Sustaining Work

First, there are the motivations to found an OSP, which we assume to be, regardless of being an organization or individual, sharing development costs and achieving widespread adoption. Hence, OSP founders must face the managerial task of attracting visitors, users and developers to create and maintain an active community that improves the application and its source code continuously. In summary, the ultimate challenge is to sustain work towards software improvement and diffusion.

Various types of contribution can help OSS accomplish this challenge. Users can request new features and spread the word to find more users, developers can implement requested features and fix bugs, and visitors can report broken links and, as readers of source code, make design suggestions, for example. Besides that, visitors, users and developers can trigger network externalities that increase project visibility and thus the general likelihoods of receiving contributions and finding new users. Accordingly, the understanding of why and how these intertwined contributions come about is crucial. In fact, a great deal of research aimed at that.

In general, we have learnt from the literature that: the type of license chosen and the presence of sponsors influence user and developer attraction as well as their intention to contribute; that organizations prefer less restrictive licenses and get involved when their business model depends on the application; and that being paid to develop OSS leads to above-average contribution levels, whereas intrinsic motivations have no detectable effect on levels of contribution (Stewart et al., 2006; Roberts et al., 2006; Watson et al., 2008; Sen et al., 2008; Santos Jr. et al., 2010).

In unfolding what motivates individuals and organizations to contribute, this research stream highlighted the importance of OSS to provide stakeholders with leverage in their mundane tasks and professional activities (utilitarian value), diminishing the role of ideology and other abstract reasons. Likely, ideology plays an important but limited role, perhaps being a non-sufficient reason to sustain motivation to contribute. High-quality, market-impacting OSS was not, and maybe could not have been, produced and maintained by ideological and passionate volunteers alone during their free time. Sponsorship has always emerged as vital to sustain work in open source projects. Yet, the source of the incentive to contribute has been mostly assumed to come from individuals rather than organizations.

The role of organizations in sponsoring and developing OSS is of primary importance. For instance, it has been publicly stated that 90% of Eclipse committers are paid employees of member companies, and our analysis shows that about 95% of Android's commits are signed by organizations (Google: ~80%). Similarly, over 60% of the more than 800k commits made to 367 projects hosted by Gnome are of authors explicitly associated with organizations. This preliminary analysis[2] indicates that

[2] The results are based on the analysis of 836,298 commits, from January 1997 to August 2011, available on Gnome's git repository, and of 110,640 commits, from October 2008 to August 2011, available on Android's kernel git repository.

none of Gnome's projects is free of organizational support, and that all of these 369 projects are managed by legal foundations or formal alliances. Nevertheless, to say that organizations are heavily involved in OSS is nothing new. However, the discussion of whether sponsorship is a required condition for OSS sustainability and what the consequences of this are to communities and founders, who perhaps design formal organizations in response this perception, is absent in the literature.

3 Implications for Theory and Practice

Although OSS scholars have recognized that OSP receive large amounts of contribution from organizations, they are not yet able to explain when and why OSP step away from an informal structure (Bazaar-Market) and become a jurisdiction of interorganizational relationships to accommodate these contributions, account for the rights and obligations of participants, and sustain work. Probably, this transformation is a result of seeking and securing sponsorship, which turns OSP into a coalition of agents with various (conflicting) interests that requires a formal and complex governance structure to be managed, resembling the Cathedral-Hierarchy or Network form of economic organization (Powell, 1990). Thus, as scholars, we see as timely to ask: Does this metamorphose expose the limits of the Bazaar organization, indicating when and why Markets fail and give rise to Hierarchical or Network governance structures?

Additionally, responsible for the transformation, founders of OSP strive to find external resources (sponsorship) as it signals credibility to the market, boosting adoption rates, and sustains contributors' motivations to develop source code and locate bugs. Such behavior is not inconsequential, as it later restricts the governance structure and coordination mechanisms that can be effectively applied. But do practitioners have another option? O'Mahony (2007) stated that OSP vary according to governance structure and degree of community-management, defining how we can observe that. However, how OSP came to have those structures and the conditions under which each structure is effective or required were left out of the paper. Thus, as practitioners, it is important to ask: Is the Bazaar organization capable of sustaining OSS improvement and diffusion in the long term? Under which conditions and towards which goals the Bazaar must incorporate Cathedral elements? To what extent can sponsorship be accommodated in a Bazaar-type of organization?

Acknowledgments

This research is supported by the Horizon Digital Economy Research Institute at the University of Nottingham, and the USP FLOSS Competence Center (CCSL-NAPSoL) with a grant from the USP Research Provost Office.

References

Agerfalk, P., Fitzgerald, B.: Outsourcing to an Unknown Workforce: Exploring Opensourcing as a Global Sourcing Strategy. MIS Quarterly. 32, 385-409 (2008)

Fitzgerald, B.: The Transformation of Open Source Software. MIS Quarterly. 30, 587-598 (2006)

O'Mahony, S.: The governance of open source initiatives: what does it mean to be community managed?. Journal of Management & Governance. 11, 139-150 (2007)

Powell, W.: Neither Market nor Hierarchy: Network Forms of Organization. Research in Organizational Behavior. 12, 295-336 (1990)

Roberts, J., Hann, I., Slaughter, S.: Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. Management Science. 52, 984-999 (2006)

Santos Jr., C.: Understanding Partnerships between Corporations and the Open Source Community: A Research Gap. IEEE Software. 25 (2008)

Santos Jr., C., Pearson, J., Kon, F.: Attractiveness of Free and Open Source Software Projects. 18th European Conference on Information Systems (ECIS), Pretoria (2010)

Sen, R., Subramaniam, C., Nelson, M.: Determinants of the Choice of Open Source Software License. Journal of Management Information Systems. 25, 3, 207-239 (2008)

Stewart, K., Ammeter, A., Maruping, L.: Impacts of license choice and organizational sponsorship on user interest and developer activity in open sources software projects. Information Systems Research. 17, 2, 126-144 (2006)

Watson, R., Boudreau, M., York, P., Greine, M., Wynn Jr., D.: The Business of Open Source. Communications of the ACM. 51, 41-46 (2008)

West, J., O'Mahony, S.: The Role of Participation Architecture in Growing Sponsored Open Source Communities. Industry & Innovation. 15, 2, 145-168 (2008)

Managing Open Source Legality Concerns – A Sustainability Catalyst

Alexander Lokhman, Salum Abdul-Rahman, Antti Luoto, Imed Hammouda
Department of Software Systems
Tampere University of Technology
Tampere, Finland
firstname.lastname@tut.fi

Abstract. As more and more software companies are integrating different Free/Libre and open source software (FLOSS) components in their products, it became more probable that a single software solution uses numerous licenses. Mixing together different open source and proprietary licenses may lead to legality complications as different licenses introduce different privileges and requirements on the use of the composed code. In this paper, we address the multi-facets of the legality concerns of open source. We further propose an open tool architecture to address such concerns.

1 Introduction

Over the last decades, Free/Libre and Open Source Software (FLOSS) has emerged as one of the most important phenomena in software engineering. In this trend, more and more companies are putting FLOSS at the center of their business strategies. Although there are many benefits to going open source, companies need to be aware of the risks associated with FLOSS. One of such risks is the legal obligations that both consumers and producers of FLOSS need to fulfill. Unfortunately, for many companies, software developers are still unaware of these issues. This may cause trouble to the corresponding companies, especially in the absence of legal departments and external legal consultants.

In this position paper, we address the various facets of open source legality compliance, arguing that the legal risks of open source have a critical influence on the sustainability of the open source movement as a whole. We further argue that handling the legality risks through shared knowledge bases and automated tools may boost the adoption of open source. Towards the end of the paper we briefly present an open tool architecture for open source legality compliance.

2 Legality Tension of FLOSS intensive systems

When addressing the legality compliance issue of FLOSS intensive systems, there are a number of factors that must be taken into account. These factors not only stem from the nature and terms of the licenses themselves, but also are related to the way the subject software is implemented, packaged, and deployed.

There are plenty of licenses and license models. A straight forward observation when working with open source licenses is that there are many of them. The Open Source Initiative [OSI] lists about 70 licenses. Popular licenses include the GNU General Public License (GPL), the Lesser GNU General Public License (LGPL), the Apache license, the Massachusetts Institute of Technology license (MIT), and the Berkeley Software Distribution license (BSD). The terms of different licenses vary considerably. To give an example, some licenses such as MIT are classified as permissive, granting very broad rights to licensees and allowing almost unlimited use of the licensed code. Other licenses such as GPL are classified as strong copyleft, requiring that works based on the licensed code be published and relicensed to others on the same terms of the initial license. In the middle are weak copyleft licenses such as LGPL, which is a compromise between permissive and strong copyleft. The LGPL grants flexibility to users when linking to licensed software libraries. However, any modifications to the original library should be contributed back on the same terms of the license. Moreover, some licenses have several versions, and there are subtle changes between different versions. A good example is the case of GPL v2 and GPL v3. In addition, the list is by no means complete, and new licenses can be introduced if so desired. For example, a new license can add some minor differences to an earlier one, thus generating a discrepancy between the licenses, or a completely new license can be introduced.

Licenses can be conflicting [Ham10]. To give an example of possible legal incompatibilities between software components, Table 1 presents a number of open source licenses and their compatibility properties (across open source components themselves) categorized into three cases: mixing and linking is permissible, only dynamic linking is permissible, and completely incompatible.

Table 1. Example Open Source Licenses and their Compatibility

	PHP	Apache	IPL	SSPL	Artistic
GPL	3	3	3	1	3
LGPL	2	2	2	1	2
BSD	1	1	1	1	1

1- Mixing and linking permissible

2- Only dynamic linking is permissible

3- Completely incompatible

As an example, a software component under the terms of GPL cannot be directly linked with another under the terms of the Apache license. In this case, the main reason is that GPL'ed software cannot be mixed with software that is licensed under the terms of a license that imposes stronger or additional terms, in this case the Apache license. The Apache 2.0 license allows users to modify the source code without sharing modifications, but they must sign a compatibility pledge promising not to break interoperability.

Is it derived or combined work? When integrating third party open source components, possibly together with own work, the restrictions and obligations which the used licenses impose may depend on whether the work is considered

as derived (derivative) or combined (collective) [Ger09]. A simple example of derived work is a modified version of the original software. However, the distinction between derived and combined works becomes trickier when producing new work by combining or linking multiple software components, possibly distributed under the terms of different licenses. Take the example of a software system *S* which is the result of linking together an open source component *C1* and an own developed component *C2*. A common interpretation is that system *S* is considered to be derived work if *C1* and *C2* link statically (linked during compile or build time) and that S is considered to be combined work if *C1* and *C2* link dynamically (the two libraries are loaded into a client program at runtime). In a typical case, however, only a judge in a court of law can make the final decision. As a matter of fact, the court decision might depend on the specific legal framework of the jurisdiction in which the case arises.

There are thousands of open source components with different risk levels depending on their usage scenario. The number of open source components has grown at an exponential rate during the last decade. This has given software developers a jump on creating software based on existing code. However, many companies are reluctant to use open source software due to the legal risks associated with the use of those components. There have been attempts to classify open source components according to their risk level [Wil10]. Table 2 gives an example categorization. Four usage scenarios are identified: using the component as a redistributable product, as part of service offering, as a development tool, and for internal use. Three levels of risks have been proposed.

Table 2. Example Software Components and their Risk Level

Component	License	Redistribution	Service offering	Development tool	Internal use
Agent++	Agent++ license	3	3	2	1
SwingX	LGPL	3	3	3	3
Libxml2	MIT	1	1	1	1
Cglib	Apache	2	1	1	1

(1) Valid (2) Possible risk (3) Clear risk

According to the authors of [Wil10], valid means that the package can be used as instructed and that no risk has been identified. Possible risk means an interpretation question has been found. This type of issues can be solved by either 1) removing/replacing the problematic files or 2) acquiring additional permissions from the respective right holder or 3) not using the package at all or 4) based on the particular company's risk preferences in such project, a company could accept the risk. Legally, an interpretation question means that an eventual realizing risk would be civil law risk, e.g. monetary (not criminal). Clear risk means that a risk that cannot be interpreted in a way that would not include the risk has been found. This type of issues can be solved only by 1) removing/replacing the problematic files or 2) acquiring additional permissions from the respective right holder or 3) not using the package at all. A company

normally cannot accept this type or risk, since it means the possibility of not only civil law risks, but criminal risks.

As an example, component Agent++ can be used internally with no risk, has a possible risk when used as a development tool, but exhibits a clear risk when used as part of service offering or a redistributable product.

Open Source legality interpretations are subject to the way software is implemented, packaged, and deployed [Ham10, Mal10]. The legality requirements imposed by FLOSS licenses, such as the requirement to publish source code (i.e. the copyleft rule of GPL), may depend for instance on the interaction type of the components (data-driven versus control-driven communication). In the case of mere data exchange between components, there is no copyleft obligation as the two components are considered as separate programs. Also, the copyleft obligation of GPL does not hold if the FLOSS component (or a modified version of it) is deployed as a hosted service. However, if the hosted code is licensed under the terms of AGPL (Affero General Public License), the copyleft requirement does hold, but only in the case of user interaction with the hosted service (in contrast to service to service interaction). In addition, the copyleft requirement of GPL may not hold in case of interactions through standardized interfaces such as the use of operating system public API, in contrast to system hacks which make the two communication components strongly coupled. Finally, compatibility concerns among different licenses may be circumvented if the packaging of components is done by the user instead of building the entire system at the vendor site.

3 Towards an Open Architecture for FLOSS Compliance

The ultimate goal of this work is to design and implement a new kind of tool for addressing the various legality compliance concerns identified in the previous section.

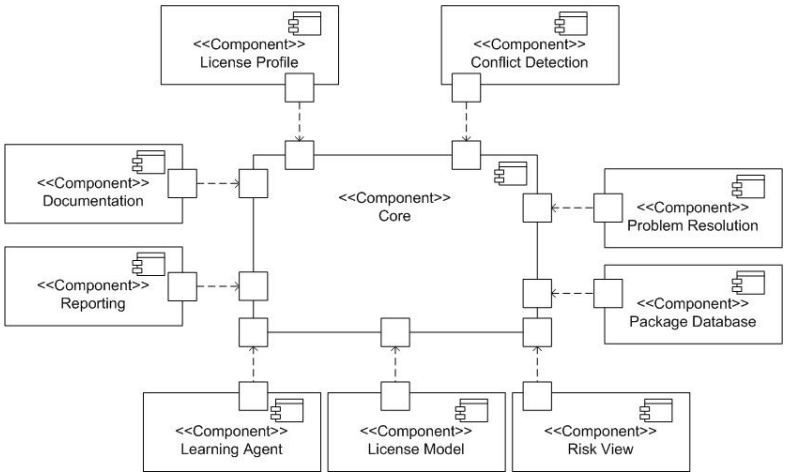

Figure 1. An Open Architecture for Open Source Compliance

Figure 1 proposes an example overall architecture for such a tool. Here we assume that the tool is capable of managing the legality concerns at the architectural level (i.e., application design is expressed as an UML component diagram for example). Table 3, in turn, explains each of the architectural components and lists example existing works that could be used as implementation guides.

Table 3. Architectural Components

Component	Description	Resource
Core	Handles interactions between the application model, licensing information and the user.	[Wil10]
License Profile	A UML extension to include license information.	[SPDX], [Hoe07], [OSI]
License Model	Describes in computable format the clauses, restrictions, rights and their interdependencies of a license.	[Als09], [Tuu09], [Hoe07], [Gom08]
Package Database	A repository of containing information on which license and copyright information is associated with which package.	[SF]
Risk View	Assess legal risks related to use of component for variable purposes re-licensing, sale, internal use etc.	[Als09], [Hoe07], [Gom08]
Conflict Detection	Analysis whether license terms of different licenses conflict when linked into the same software.	[Ham10], [Als09], [Tuu09], [FOS10], [OSLC], [Ninka]
Problem Resolution	Suggests operations that can be performed to remove license conflicts from model.	[Ger09], [Ham10], [Mal10]
Learning Agent	Records user actions so that they can be later used to improve program performance.	[Ham10]
Reporting	The analysis results from the different components can be output in different formats.	[FOS10], [Tuu09], [OSLC]
Documentation	Linking to internal and external documentation on open source licensing concerns.	[IFOSS]

A part from *Core*, each component is associated with an extension point. The architecture is made extensible so that the tool is able to work with different licenses. The *License Profile* component allows for attaching different licensing concepts to the architectural model. Different implementations of *License Model* give different interpretations of clauses based on local law. Different open source components can be registered to the tool via the *Package Database* component. The *Risk View* extension point allows the plug-in of different risk analysis methods. The tool also integrates different techniques for detecting conflicts among licensed components (*Conflict Detection*) and proposes remedial actions (*Problem Resolution*). These actions can be recorded for future exploitation (*Learning Agent*). Finally, the tool is capable to report the analysis results in different pluggable formats (*Reporting*) and links to relevant documentation resources (*Documentation*). We argue that the described

architecture allows the building of an open knowledge base related to open source licensing.

4 Conclusions

There has been a growing interest in studying the compliance of software systems with respect to the legality restrictions and obligations of open source licenses. This came in response to the increasing concerns about the legal risks of using FLOSS components. We argue that if such issues are not addressed by both legal experts and software developers, the whole open source ecosystem may face sustainability challenges. In this paper we have presented an overview of the main dimensions involved in open source compliance. Based on the analysis, we have outlined an open architecture for managing open source legality concerns at the architectural level. As future work, we plan to exploit the ideas presented in this paper to develop concrete tool infrastructure.

References

[Als09] Alspaugh, T. A., Asuncion, H. U. and Scacchi, W. Analyzing Software Licenses in Open Architecture Software Systems. In proc. of FLOSS 2009, pp 54-57.

[FOS10] FOSSology. http://fossology.org/. Last accessed Sep. 2011.

[Gam05] Gamma, E.,Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison_Wesley, 1995.

[Ger09] German, D. M.; Hassan, A. E. License integration patterns: Addressing license mismatches in component-based development. In proc. of ICSE 2009, pp 188-198. May 2009.

[Ham10] Hammouda, I., Mikkonen, T., Oksanen, V. and Jaaksi, A. Open Source Legality Patterns: Architectural Design Decisions Motivated by Legal Concerns. In proc. of AMT 2010. Tampere, Finland. October 2010. ACM Press.

[Hoe07] Hoekstra, R., Breuker, J., Di Bello, M. and Boer, A. The LKIF Core Ontology of Basic Legal Concepts . In proc. of LOAIT 2007, pp 43-63.

[IFOSS] International Free and Open Source Software Law Review. http://www.ifosslr.org. Last accessed Sep. 2011.

[OSI] Open Source Initiative. http://www.opensource.org. Last accessed Sept. 2011.

[OSLC] OSLC, Open Source License Checker. http://sourceforge.net/projects/oslc. Last accessed Sept. 2011.

[Mal10] Malcolm, B. Software Interactions and the GNU General Public License. IFOSS L. Rev, 2(2), pp 165 - 180. 2010.

[Ninka] Ninka, a license identification tool for Source Code. http://ninka.turingmachine.org/. Last accessed Sep. 2011.

[SF] Sourceforge.net. http://sourceforge.net/. Last accessed Sep. 2011.

[SPDX] Software Package Data Exchange (SPDX). http://spdx.org/. Last accessed Sep. 2011.

[Tuu09] Tuunanen, T., Koskinen, J. and Kärkkäinen, T. Automated software license analysis. Automated Software Engineering 16 (3-4), 455-490, Dec. 2009.

[Wil10] von Willebrand, M. and Partanen, M. P. Package Review as a Part of Free and Open Source Software Compliance. IFOSS L. Rev, 2(2), pp 39 – 60. 2010.

[Gom08] Gomez, F. P. and Quiñones, K. S. Legal Issues Concerning Composite Software. In proc. of ICCBSS 2008, pp 204-214, 2008.

Open Source communities for long-term maintenance of digital assets: what is offered for ODF & OOXML?

Jonas Gamalielsson and Björn Lundell

University of Skövde, Skövde, Sweden,
{jonas.gamalielsson, bjorn.lundell}@his.se

1 Introduction

Many organisations have requirements for long-term maintenance of their software systems and digital assets. In this paper, we explore sustainability of Open Source Software (OSS) licensed office applications (hereafter referred to as OSSOA) implementing the ISO-standardised XML-based document formats ODF (ISO/IEC 26300:2006) and OfficeOpen XML (ISO/IEC 29500:2008). We draw from prior research conducted in the Swedish public sector context where different applications for the two formats are used (Lundell 2011, Lundell & Gamalielsson 2011).

In a number of countries there are governmental policies mandating use of ODF in the public sector (Lundell 2011). Previous research in the Swedish public sector found that "there is little or no evidence of consideration given to document formats when municipalities procure software" (Lundell 2011). The same study found significant confusion concerning the concept of standard and differences between file formats and software applications.

Implementations of file formats that are distributed under an Open Source license clearly contribute to the desired economic effect of standards (FLOSSPOLS 2005), in that such implementations stimulate competition in the marketplace and minimise the risk for different types of lock-in effects (Lundell 2011). For example, organisations have had concerns for avoiding vendor lock-in for decades (Guijarro 2007).

Different Open Source licenses have different effects on longevity of tools, and licence selection is a critical issue for companies and communities (Engelfriet 2010). Previous results on Open Source implementations of document formats show that community commitment and choice of licenses may significantly affect long-term maintenance of digital artefacts (Lundell & Gamalielsson 2011). Further, company commitment and choice of software licenses affect longevity of tool support for different file formats (Lundell et al. 2011).

Before an organisation adopts an Open Source project it is important to evaluate its communities in order to make sure that it is healthy and that the project is likely to be sustainable and maintained for a long time (van der Linden et al. 2009). From this, our goal is to investigate the availability of effective and sustainable OSSOAs for creation and editing of documents.

2 Research approach

Firstly, we identify effective and sustainable OSSOAs implementing the two XML-based formats ODF and Office Open XML (OOXML). Hence, we undertook: a

systematic investigation of existing OSSOAs through a literature analysis, interviews, search in OSS forges, and an analysis of information on existing OSS available through framework agreements provided by the Legal, Financial and Administrative Services Agency (*swe.* Kammarkollegiet). Our analysis identified several OSSOAs for creation and editing of documents in the ODF format, whereas we only identified one OSSOA that provides support for creation and editing of documents in the OOXML format (docx4all). For the OOXML format, our selection of docx4all for investigation was informed by a previous analysis covering OSSOAs that provide support for creation and editing of documents (Garshol, 2010). We note that our study covers all tools using ".docx" as its internal format (although the ECMA, instead of the ISO version). It was decided to explore the LGPL-licensed LibreOffice (LO) project as a representative of an effective OSSOA implementing the ODF format. Our review revealed that both LO and OpenOffice.org (OO) are in professional use with the ODF format in the Swedish public sector, whereas there is no OSSOA in professional use for the OOXML format. For the study we selected LO as an analysis would encompass an investigation of consequences of a fork. Further, amongst organisations already using OO we observed an interest in exploring LO as a potential alternative. We establish the sustainability for the selected Open Source projects through an assessment of contributions to the Software Configuration Management system[1] (SCM) over time.

Secondly, for each document format for which there exists an Open Source implementation with a sustainable community, our goal was to extend the analysis with a deeper characterisation of the Open Source community. For the ODF format, we analysed the community of the LO project in order to reveal its potential for long-term maintenance. This project constitutes a fork from the OO project. Our investigation addresses both the base project (OO) until the fork and the branch project (LO) after the fork. Specifically, we investigated the LO developer community over time with the view to obtain insights concerning long-term sustainability by means of a quantitative analysis. In so doing, we extend a previous analysis of the OO developer community (Meeks 2008) by elaborating on the effects of the fork.

To investigate the sustainability of Open Source communities, we adopt the approach from an earlier study (Gamalielsson et al. 2011) in order to analyse the contributions in terms of committed SCM artefacts of the Open Source projects over time. The data for the LO project was collected from the LO website (www.libreoffice.org/get-involved/developers, accessed 8 Aug. 2011), and for docx4all the data was collected from the docx4all website (dev.plutext.org/trac/docx4all, accessed 8 Aug. 2011).

3 Results

The developer activity in the selected OSSOAs is presented in Figure 1, which shows the number of commits for each month over the four year time period from August 2007 to July 2011 for LO (blue trace) and docx4all (red trace). Our SCM analysis of the LO project includes the development in OO before the fork in September 2010. We note that activity in the LO project varies, with a distinct peak in connection with the OO 2.4 release in March 2008 (59804 commits in April, which is not visible in the diagram for scaling reasons). Since October 2008 (with the release of OO 3.0) there

[1] Examples include Git, SVN and CVS.

have been 2851 commits each month on average. We also note that the activity is much lower in docx4all (red trace in Figure 1), and that there has been very limited SCM activity in the project since September 2009. Further, there have been no contributions to the SCM since September 2010 (when only two commits were provided) except for a single commit in April 2011. The maximum number of commits in docx4all during one month is 195 (during May 2008). Since the start of the project (October 2007) there have been 29 commits each month on average. We acknowledge that docx4all is a much smaller project than LO, and therefore it is not unexpected that docx4all has had fewer commits. Totally, there have been 664 committers (unique IDs in the SCM log) in the LO project over the studied four year period, whereas only two committers have contributed to docx4all since the creation of the project (where one of these committers has contributed 79% of all commits).

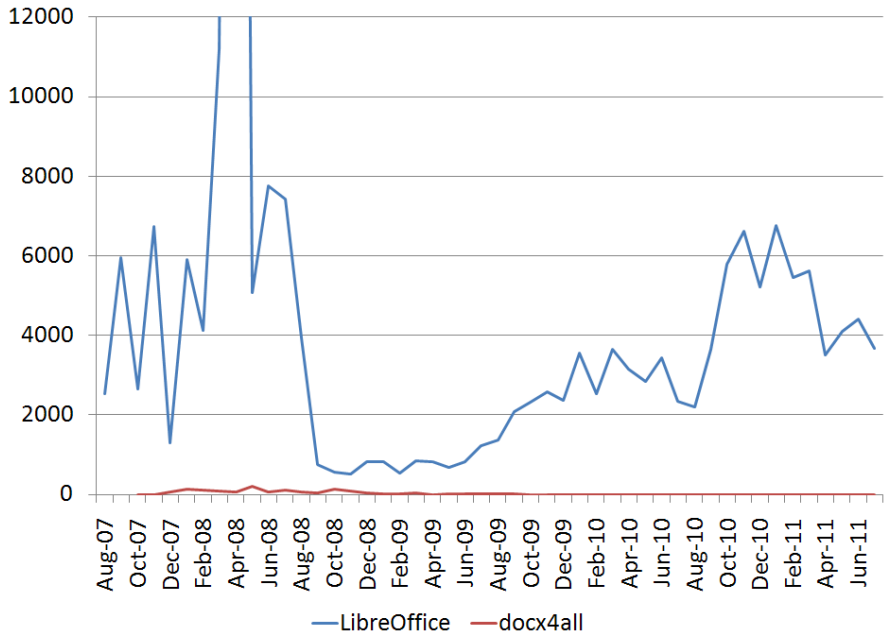

Fig 1. Number of commits over time for LO (blue) and docx4all (red)

Due to the absence of a sustainable developer community for docx4all, we decided to only consider LO in our further analysis. The proportion of commits for the 10 most active affiliations over time in the LO project is shown in Figure 2 (like in Figure 1, the peak in April 2008 is not visible for scaling reasons). It can be observed that "openoffice" is dominating until August 2010, and that other affiliations break the dominance from September 2010 (the month of the fork) and onwards. It is also noted that "sun" is most active in the period from October 2009 to July 2010, and that "oracle" is most active from August 2010 to March 2011. Further observations are that "novell" and "suse" have been active for the entire four year period with an increased activity after the fork, and that "redhat" has become the major contributor ever since the fork.

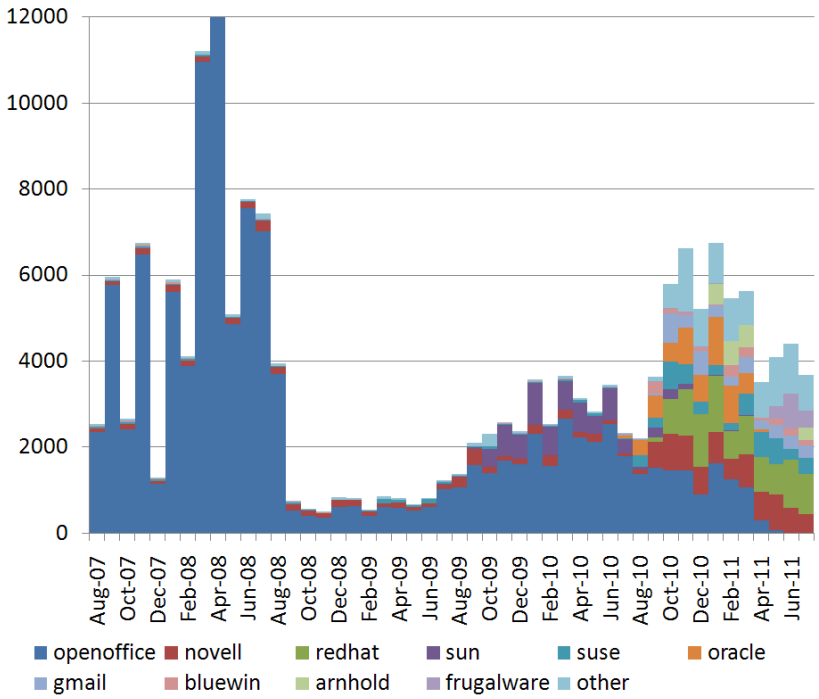

Fig 2. LO: Proportion of commits per affiliation over time

Figure 3 illustrates the total affiliation commit influence for LO 10 months before and after the fork on 28 September 2010, and further emphasizes the shift from "openoffice" domination to a more diversified developer community after the fork.

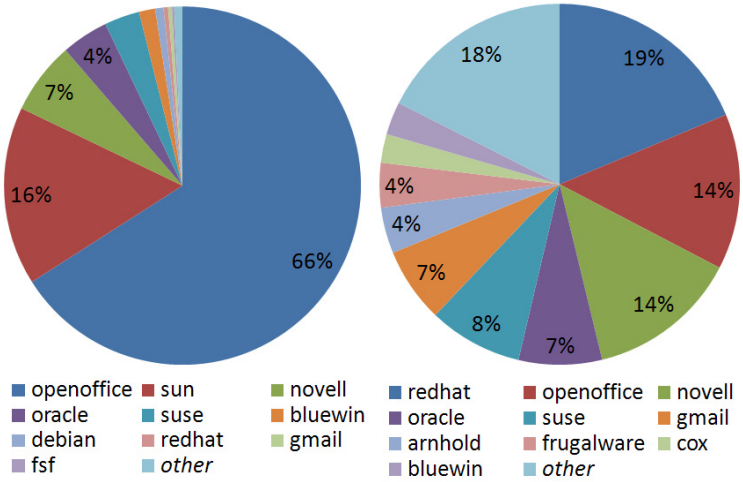

Fig 3. LO: total affiliation commit influence (left pie: during 10 months before the fork, right pie: during 10 months after the fork)

4 Conclusion and discussion

We find that there is a sustainable and effective OSSOA for the ODF document format but not for OOXML. Further, despite the relatively short time window after the fork, our analysis indicates that the LO project has an active and diversified developer community. This is in contrast with worrying observations made three years earlier for the OO project in which Meeks (2008) claims that the OO project "is a profoundly sick project, and worse one that doesn't appear to be improving with age." However, we acknowledge the inherent complexity in quantitatively analysing the OO and LO projects for a number of reasons, such as governance models and differences in practices regarding use of affiliation during project interactions.

Long term maintenance of digital assets and supporting OSSOAs is a significant issue for both private and public sectors. It is closely related to longevity of file formats. In a number of usage scenarios it is common that organisations need to preserve their systems and associated digital assets for more than 30 years (Lundell et al. 2011, Lundell 2011). It is important to recognise that any action in the private and public sectors must be based on a long-term vision and the recognition that no provider will remain on the market for the life-span during which digital assets must be maintained. For achieving a long-term sustainable ICT ecosystem it is important to recognize that for each file format used in the public sector there must exist a supporting Open Source implementation. Further, before a file format is used and referred in a public sector procurement, we argue that there should exist a sustainable implementation that is licensed under a "share-alike" or "keep-open" Open Source license. Such a requirement would be one effective strategy for minimising the risk for vendor lock-in and lack of interoperability in the public sector. It is an open question to what extent the Document Foundation may establish long-term sustainability for LO with its use of a "keep-open" license.

In acknowledging that our research has not addressed the extent to which the specification of each ISO standard is actually implemented in an OSSOA, we note that for the OOXML standard it has been claimed that "it is unclear whether anyone is able to implement the ISO standard" (documentfoundation.org 2011). Further, at time of writing, we note that there is uncertainty concerning when support for this standard will be provided by an office application[2] under any software license.

For further work, we plan to broaden our investigation of long-term sustainable communities to include both branches of the OO project, and also extend our analysis of sustainability and commitment of individuals to the projects over time. In addition, we plan to undertake a qualitative analysis of the projects, including investigation of: external events, choice of license, foundations, governance and work practices. A related issue for further investigation concerns the extent to which it is possible to successfully migrate documents between implementations of the two document formats. It is not uncommon that office applications provide warnings when a user attempts to save a document in a non-native file format. For example, in addition to its native ODF support, the most recent release (version 3.4.2) of the office application LibreOffice also provides options for saving a document in several non-native file formats. When attempting to save a document using the "Office Open XML Text" option in the office application LibreOffice, the application issues the warning: "This document may contain formatting or content that cannot be saved in Office Open XML Text file format. Do you want to save the document in this format anyway? Use the

[2] In fact, it has been estimated that full support for creation and editing of documents in OOXML will be provided in a proprietary office application 'no later than Office "15."' (Mahugh 2010).

latest ODF file format and be sure all formatting and content is saved correctly." (documentfoundation.org 2011). Such warnings may cause users to vary and the extent to which successful migration between formats actually is (and can be) provided during long life-cycles beyond any single office application needs further exploration.

Acknowledgement

This research has been financially supported by the ITEA2 project OPEES (www.opees.org) through Vinnova (www.vinnova.se).

References

documentfoundation.org (2011). LibreOffice OOXML, The Document Foundation Wiki, http://wiki.documentfoundation.org/LibreOffice_OOXML, accessed 15 September 2011.

Engelfriet, A. (2010). Choosing an Open Source License, IEEE Software, 27(1): 48-49

FLOSSPOLS (2005). An Economic Basis for Open Standards, Deliverable D4, December 12, 2005, http://flosspols.org.

Gamalielsson, J., Lundell, B. and Mattsson, A. (2011). Open Source Software for Model Driven Development: A Case Study, In Hissam, S. (Eds.) Open Source Systems: Grounding Research, IFIP Advances in Information and Communication Technology, Vol. 365, ISBN: 978-3-642-24417-9, Springer, Boston, pp. 348-367.

Guijarro, L. (2007). Interoperability frameworks and enterprise architectures in e-government initiatives in Europe and the United States. Government Information Quarterly, 24 (1), 89-101.

van der Linden, F., Lundell, B. and Marttiin, P. (2009). Commodification of Industrial Software: A Case for Open Source. IEEE Software, 26 (4): 77-83.

Lundell, B. (2011). e-Governance in public sector ICT-procurement: what is shaping practice in Sweden?, European Journal of ePractice, 12(6), http://www.epractice.eu/en/document/5290101

Lundell, B. & Gamalielsson, J. (2011). Towards a Sustainable Swedish e-Government Practice: Observations from unlocking digital assets. In Proceedings of the IFIP e-government conference 2011(EGOV 2011), Delft, The Netherlands, 28 August - 2 September 2011.

Lundell, B., Gamalielsson, J. & Mattsson, A. (2011). Exploring Tool Support for Long-term Maintenance of Digital Assets: a Case Study, In Fomin, V. and Jakobs, K. (Eds.) Proceedings: 16th EURAS Annual Standardization Conference, European Academy of Standardisation, The EURAS Board, pp. 207-217.

Mahugh, D. (2010). Office's Support for ISO/IEC 29500 Strict, http://blogs.msdn.com/b/dmahugh/archive/2010/04/06/office-s-support-for-iso-iec-29500-strict.aspx, accessed 15 September 2011.

Meeks, M. (2008). Measuring the true success of OpenOffice.org, http://people.gnome.org/~michael/blog/ooo-commit-stats-2008.html, accessed 15 September 2011.

Adding Control to Open Innovation Projects Through Agile Practices

Terhi Kilamo[1], Ville Kairamo[2], Petri Räsänen[2], and Jukka P. Saarinen[3]

[1] Tampere University of Technology firstname.lastname@tut.fi
[2] Uusi Tehdas/New Factory firstname.lastname@hermia.fi
[3] Nokia Research Center jukka.p.saarinen@nokia.com

Abstract. Businesses today have to rely on rapid development and release cycles. Thus open innovation has emerged as an increasingly appealing option also for the software business to gain variant ideas and concepts. A local open innovation platform for students, Demola, allows university students to work on real life industrial cases of their own interest. We monitored the daily work routine of a student team and found that practises from agile software development were applied to scope and manage the project activities.

1 Introduction

Many companies rely on innovation on a daily basis to create better products and to improve their internal processes [2] Constant, lightning-fast innovation is becoming an essential element of product development also in the software business. Open innovation helps in identifying the best ideas by combining internal and external ideas [7, 2].

Iterative or agile software development [6] has become more popular over the more traditional processes in the software industry. Agile development practises, mainly the concept of sprinting, has been studied earlier in the context of free software [1]. In this paper we focus on the agile approach in the open innovation setting. Innovation work is similarly characterized by ideas, changes and do as you go attitude. The absence of formal processes and excess documentation is characteristic to it in accordance to the manifesto for agile software development [5] with emphasis on interaction, collaboration and change when necessary.

The paper focuses on open innovation in the context of academia-industry cooperation in the form of a local open innovation platform Demola [8]. One of the aims of the platform is to develop an open innovation environment that is multidisciplinary and agile in the sense that innovations can flow freely and are not restricted to any artificial process or framework that must be obeyed in order to benefit from it. In this paper we discuss how practices of agile software development can be incorporated to compensate such innovation challenges as timely delivery, communication, and quality. We have conducted our study by interviewing the key people behind the environment and by observing an

example development team for identifying their working practices. The main research question was: What kind of development practises are used to work on the projects and how do they compare to agile practises?

The rest of the paper is structured as follows. Section 2 motivates the work by introducing the open innovation platform, Demola. Section 3 discusses the practices agile development in the open innovation context and further highlights them in practise through an example team. The results of the paper are discussed in Section 4 and finally Section 5 concludes the paper with some final remarks.

2 Platform for Open Innovation and Learning

There is a real need for increased opportunities for innovation projects that can lead to new business ideas. Open innovation environments allow businesses to reach beyond the company scope in the search for new concepts and ideas. A local open innovation platform, Demola, provides a governance framework needed with practices and working principles to bring innovation partners together and to ensure ongoing innovation work.

2.1 Demola Organisation

Demola is a modern learning environment for students from different universities. It aims to multidiciplinary and agile development of innovative products and demos. The project ideas come from the industry and public organisations and thus concepts that have practical business importance are developed. The student work is supported by both the industrial and the academia partners, who provide guidance throughout the project. Figure 1 shows the partners in Demola and the flow of communication and support for the project work.

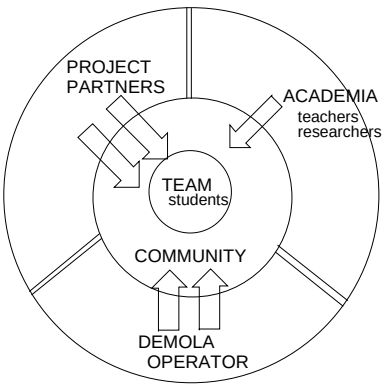

Fig. 1. Demola Partners

Demola offers a governance framework that facilitates team building and supports emerging business ideas. It also incorporates a model for managing immaterial rights that supports startups and respects the authors. On a practical level, Demola provides workspaces that support team work and co-creation.

3 Incorporating Agile Development Practices

Projects come to Demola through industrial project partners and therefore they have an intended outcome, no matter how loosely defined, and a fixed timeframe. The teams are also rather small in size and new members are normally not added after project kickoff. Iterative or agile software development that has gained popularity in the software industry over the more traditional processes lends a way to handle the innovation projects. There are several agile approaches available, such as Scrum [6, 3] or Extreme Programming [6, 4]. We focus here on practices general to the idea of agile and not to any specific approach.

In [6] the authors find that software development can be said to be agile when the releases are small but done often, the customer and the developers work together and in close communication, the development method is straightforward and adapts to the situation making it easier to do rapid changes. These are all identifiable in Demola as the independent teams appear to control their development cycle through applying practices known from agile development. This also in part aids Demola in being a sustainable open innovation community through completed projects.

Rapid Release Cycle Innovative development starts from ideas and concepts. An ideal project timeframe is short in Demola. A typical project is three to four months in duration. Development is done in small increments, the final outcome is loosely specified and the teams have a lot of fluidity in the specification. The current state of the project is demoed regularly to the customer. An agile, demo-driven development approach with frequent demos enables control of the project focus and its intended outcome.

Close Communication The teams commonly meet with eachother and the customer on a regular basis during the lifecycle of their Demola project. It is typical that teams keep in touch regularly, mostly daily, to sync their work progress via chats, online phone applications or meetings. Even though there are no product releases during the life cycle of the project the customer gets the current version of the product in these meetings. Changes can be made to the requirements and project outcome based on the teams work. While the requirements management is flexible with requirements changed and added as the project evolves, the project runs for a predetermined time. Similar fixed time projects are known from agile software development and give the project customer control over the end product. They can add, remove and prioritize the requirements as they go thus controlling the outcome of the project.

Self-Managing Teams The teams themselves can be seen through agile practises, where development is built around small development teams or pairs. One Demola project team forms such a unit and has freedom in choosing and adapting the working methods and arrangements as they see fit. There is likely to be a wide variation of practices here as the teams and projects vary. What is

common to them is the Demola workplace that provides premises and tools to enable independent, collaborative work of the teams as they best see fit.

3.1 Sample Case: Team Practices

We monitored the work of one team through the course of their development project to see how our observations on Demola apply to the daily work routine. The team was selected as it had a suitable kickoff date and project schedule and both the team and their customer project partner had no objections on us observing their collaboration. The development practices relied on iterative development with one week intervals.

There were five members in the sample case team. The educational background of the participants varied with one of the team members having completed their master's degree. One had a bachelor of science degree while three were still working on completing their undergraduate studies. The cultural background was diverse with members of four different nationalities and from two different continents. However, all participants in the project were software engineering majors even though innovation projects would benefit from a wider view with participants majoring in usability, human sciences or graphics design to name a few.

3.2 Team Collaboration

The team collaboration was informal but had certain structuring elements in it. The overall format of communication and syncing followed agile practices. There were more quiet and more talkative people in the team but no sense on dictatorship emerged. The eldest team member could be seen as a team leader and was also voted into that position by the team. They found a named leader necessary to keep the work in sync and for managing the work load.

At the beginning of the project the team decided to keep in touch daily to sync what everyone has done. As the development started in earnest the team abandonded such a strict, approach and adapted to a more flexible once a week sync. A daily sync would have followed agile methods better, but the teams self direction abandoned the approach.

3.3 Team and Customer Interaction

The team met the customer project partners weekly in a meeting at the project partners offices. A demo was prepared to show the progress that week. The length was roughly one hour, never over two. The meetings were informal but followed a certain structure that resembled a process known from agile methods. What progress the team had done during the week was discussed over a demo and what needed to get done in the future was agreed upon based on that. The possible problems, or impediments, that stood in the way of the team were covered together with if the customer could help the team in solving them was

also covered. A checklist of the project's status was maintained not only to keep track of the project but also to map the changing and emerging requirements. Both the customer and the team were able to make changes to the requirements but the customer had the final say. The customer acted as a product owner in agile.

The team members and a person responsible for the project on the customer's side were always present at the meetings. In addition, people from the customer company interested in the project attended when necessary. These outside experts were also brought in to aid in a development issue or give insight on technical topics.

4 Discussion

Demola is at heart a community. Additionally its day to day practises lean towards agile methods for managing the project as community driven development approach alone does not provide sufficient tools for timeboxing or requirements management. This brings a natural addition to the innovation work without endangering the community level principles.

The participants have the final responsibility of the work and project outcome. The teams keep in close communication not only with eachother but also with the project partner. Furthermore, frequent demos add flexibility to the requirements. Based on the overall Demola approach and the work of the sample case team, the agile approach appears as a viable way for the teams to keep the project on track and to adjust it to the needs of the project partner during the project. Ability to meet the project requirements and create innovative products and demos within Demola is an important factor in Demola's sustainability as Demola is dependent on industrial partner's project ideas.

The biggest limitation of our research on Demola so far is that our observation is limited to one example team. There is a risk that we get an overly idealistic and onesided view of the teams based on just one project. We believe the results are applicable to other teams as well since the agile practices were identified from the Demola community as well before monitoring a project team. We intend to enforce the work through a wider study of the Demola projects. Identifying further how variance in team member's background and multidiciplinary teams effects the project work is also of interest.

5 Conclusions

We set out to study an open innovation approach and how the daily workflow of the project development in a single team shares similarities with agile software development to control the innovative work flow. Our findings suggest challenges of community work can be addressed by the adoption of such new practises as time-boxing, face to face meetings, and demo-driven development.

References

1. Capiluppi A. and Adams P.J. Bridging the gap between agile and free software. *International Journal of Open Source Software and Processes*, 1(1):58–71, 2009.
2. Chesbrough H. *Open Innovation: Researching a New Paradigm*, chapter Open Innovation: A New Paradigm for Understanding Industrial Innovation. Oxford University Press, 2006.
3. Takeuchi H. and Nonaka I. The New New Product Development Game. *Harvard Business Review*, pages 137–146, January-February 1986.
4. Beck K. Embracing Change With Extreme Programming. *Computer*, 32(10):70–77, October 1999.
5. Beck K., Beedle M., van Bennekum A., Cockburn A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R.C., Mellor S., Schwaber K., Sutherland J., and Thomas D. Manifesto for Agile Software Development. Available at: `http://agilemanifesto.org/`, March 2002. Last visited March 2011.
6. Abrahamsson P., Salo O., Ronkainen J., and Warsta J. Agile Software Development Methods Review and Analysis. VTT Publications 478, 2002.
7. Davis S. How to Make Open Innovation Work in Your Company. *Visions Magazine*, December 2006.
8. Demola Innovation Platform. `http://www.demola.fi`. Last visited March 2011.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland