IFIP AICT 451

Ernesto Damiani
Fulvio Frati
Dirk Riehle
Anthony I. Wasserman
(Eds.)

Open Source Systems: Adoption and Impact

11th IFIP WG 2.13 International Conference, OSS 2015
Florence, Italy, May 16–17, 2015
Proceedings

Springer

IFIP Advances in Information and Communication Technology 451

Editor-in-Chief

Kai Rannenberg, Goethe University Frankfurt, Germany

Editorial Board

IFIP – The International Federation for Information Processing

IFIP was founded in 1960 under the auspices of UNESCO, following the First World Computer Congress held in Paris the previous year. An umbrella organization for societies working in information processing, IFIP's aim is two-fold: to support information processing within its member countries and to encourage technology transfer to developing nations. As its mission statement clearly states,

> *IFIP's mission is to be the leading, truly international, apolitical organization which encourages and assists in the development, exploitation and application of information technology for the benefit of all people.*

IFIP is a non-profitmaking organization, run almost solely by 2500 volunteers. It operates through a number of technical committees, which organize events and publications. IFIP's events range from an international congress to local seminars, but the most important are:

- The IFIP World Computer Congress, held every second year;
- Open conferences;
- Working conferences.

The flagship event is the IFIP World Computer Congress, at which both invited and contributed papers are presented. Contributed papers are rigorously refereed and the rejection rate is high.

As with the Congress, participation in the open conferences is open to all and papers may be invited or submitted. Again, submitted papers are stringently refereed.

The working conferences are structured differently. They are usually run by a working group and attendance is small and by invitation only. Their purpose is to create an atmosphere conducive to innovation and development. Refereeing is also rigorous and papers are subjected to extensive group discussion.

Publications arising from IFIP events vary. The papers presented at the IFIP World Computer Congress and at open conferences are published as conference proceedings, while the results of the working conferences are often published as collections of selected and edited papers.

Any national society whose primary activity is about information processing may apply to become a full member of IFIP, although full membership is restricted to one society per country. Full members are entitled to vote at the annual General Assembly, National societies preferring a less committed involvement may apply for associate or corresponding membership. Associate members enjoy the same benefits as full members, but without voting rights. Corresponding members are not represented in IFIP bodies. Affiliated membership is open to non-national societies, and individual and honorary membership schemes are also offered.

More information about this series at http://www.springer.com/series/6102

Ernesto Damiani · Fulvio Frati
Dirk Riehle · Anthony I. Wasserman (Eds.)

Open Source Systems: Adoption and Impact

11th IFIP WG 2.13 International Conference, OSS 2015
Florence, Italy, May 16–17, 2015
Proceedings

Springer

Editors
Ernesto Damiani
Department of Information Technology
Università degli Studi di Milano
Crema
Italy

Fulvio Frati
Computer Science Department
Università degli Studi di Milano
Crema
Italy

Dirk Riehle
Department Informatik
Friedrich Alexander University
Erlangen-Nürnberg
Erlangen
Germany

Anthony I. Wasserman
Carnegie Mellon University
Moffett Field, CA
USA

ISSN 1868-4238 ISSN 1868-422X (electronic)
IFIP Advances in Information and Communication Technology
ISBN 978-3-319-17836-3 ISBN 978-3-319-17837-0 (eBook)
DOI 10.1007/978-3-319-17837-0

Library of Congress Control Number: 2015936520

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

Preface

Welcome to the 11th International Conference on Open Source Systems, marking 10 years since the initial OSS conferences. This annual conference is now recognized as the primary event for the open source research community, attracting not only high-quality papers, but also building a community around the technical program, a collection of workshops, and a Doctoral Consortium. We made several changes in planning the technical program this year. First, we sharply reduced the size of the Program Committee and added people who had not previously served on as an OSS Program Committee. That step made it easier for the Program Committee members and the Program Committee chairs to calibrate the reviews. We were very pleased to receive 50 submissions for the technical program, from which we selected 15. We are most grateful to the Program Committee members for handling the larger than average reviewing load, and for their thoughtful reviews, all done on a tight schedule.

We chose the title "Open Source Systems: Adoption and Impact" for this volume in recognition of the change in the role of open source software within companies and organizations. When the OSS conferences started in 2005, open source software was often treated as a "second-class" citizen, with many organizations claiming (incorrectly in most cases) not to use open source and its advocates viewed as being out-of-touch with "real world" software development and deployment issues. Now, however, open source software is fully mainstream. Virtually all of the major software and systems companies are active users of open source, and are often contributors as well. Companies have established their own policies for use of open source, and routinely include open source in their products. Linux servers are common, while organizations rely on Hadoop and other open source tools for managing large volumes of data. Most importantly, open source software has moved from its early status of providing functional equivalents of proprietary software to today's situation where developers and researchers are innovating with open source software. In key software areas such as data management, cloud infrastructure, service-oriented architectures, mobile operating systems, and the Internet of Things, open source software solutions are frequently among the leaders in their category. Even so, proprietary software still has a major role, and will retain that role in the foreseeable future. Organizations and institutions still have massive existing investments in proprietary software, with many disincentives for moving away from those products. However, as organizations look for ways to save money, open source software becomes increasingly attractive, particularly as potential users gain greater confidence in its quality and security. The open source research community is helping to drive this transition, working with organizations to help them evaluate and adopt open source, often based on research studies published in the OSS

conference series. We are pleased to contribute this volume of papers to the growing body of knowledge about open source software. The research results presented in this conference can have a significant influence on the future of open source software.

May 2015

Anthony I. Wasserman
Dirk Riehle
Ernesto Damiani

Organization

Organizing Committee

General Chair

Ernesto Damiani	Università degli Studi di Milano, Italy

Program Chairs

Dirk Riehle	Friedrich-Alexander-University Erlangen-Nürnberg, Germany
Anthony I. Wasserman	Carnegie Mellon University, USA

Ph.D. Contest Chair

Slinger Jansen	Utrecht University, The Netherlands

Regional Publicity Chairs

Moataz Ahmed	King Fahd University of Petroleum and Minerals, Saudi Arabia
Scott Hissam	Software Engineering Institute, Carnegie Mellon University, USA
Karl Reed	La Trobe University, Australia
Francesco Zavatarelli	Università degli Studi di Milano, Italy

Organizing Chairs

Nadia Fusar Poli	Università degli Studi di Milano, Italy
Fulvio Frati	Università degli Studi di Milano, Italy

Finance Chair

Fulvio Frati	Università degli Studi di Milano, Italy

Program Committee

Chintan Amrit	University of Twente, The Netherlands
Luciano Baresi	DEIB - Politecnico di Milano, Italy
Paolo Ciancarini	University of Bologna, Italy
Francesco Di Cerbo	SAP Research Sophia-Antipolis, France
Jonas Gamalielsson	University of Skövde, Sweden
Jesus M. Gonzalez-Barahona	Universidad Rey Juan Carlos, Spain
Imed Hammouda	Chalmers University of Technology and University of Gothenburg, Sweden
Abram Hindle	University of Alberta, Canada
Netta Iivari	University of Oulu, Finland
Stefan Koch	Boğaziçi University, Turkey
Fabio Kon	University of São Paulo, Brazil
Luigi Lavazza	Università degli Studi dell'Insubria, Italy
Eda Marchetti	ISTI-CNR, Italy
Audris Mockus	University of Tennessee, Knoxville, USA
Sandro Morasca	Università degli Studi dell'Insubria, Italy
John Noll	Lero - the Irish Software Engineering Research Centre, University of Limerick, Ireland
Mauro Pezzè	University of Lugano, Switzerland
Stephane Ribas	Inria, France
Gregorio Robles	Universidad Rey Juan Carlos, Spain
Steve Schmid	Open Technology Foundation, Australia
Alberto Sillitti	Free University of Bozen-Bolzano, Italy
Diomidis Spinellis	Athens University of Economics and Business, Greece
Megan Squire	Elon University, USA
Klaas-Jan Stol	Lero - the Irish Software Engineering Research Centre, University of Limerick, Ireland
Giancarlo Succi	Free University of Bozen-Bolzano, Italy
Davide Tosi	University of Insubria, Italy
Aaron Visaggio	University of Sannio, Italy
Stefano Zacchiroli	Université Paris Diderot, France

Sponsored by

Supported by

UNIVERSITÀ
DEGLI STUDI
DI MILANO

Keynote Talks

Building a Commercial Open Source Software Company

Paul Fremantle

School of Computing, University of Portsmouth, Portsmouth, UK
paul.fremantle@port.ac.uk

Abstract. This paper is based on personal experiences in building a venture-capital backed Open Source company, starting in 2005. WSO2 is now a company with hundreds of customers including major brands like eBay, Boeing, Fidelity, Trimble, UBS, and many others. For example, eBay's systems running WSO2's servers handle more than 6 billion requests per day. In that time there has been a significant shift in the business models, approaches, funding and valuations of Open Source Software companies.

1 Choosing a Business Model

The first challenge of creating a company around Open Source is choosing a business model. There are multiple options. You can create a company that mainly does support for an existing project, which is fundamentally a Professional Services company. This reduces the valuation of your company[1], but is quicker and more cost-effective to get started. However, most companies aspire to be Product companies. Ten years ago the most popular approach for this was to use the GPL license and to offer companies a more "business-friendly" proprietary license to those who would pay. This model was used by MySQL. Since then a more popular model has emerged - often called "Open Core". In this model there are two versions of the product: a "community edition" that is licensed as Open Source, and an extended version that is proprietary.

WSO2 actually did not choose any of these options: we chose to use the Apache License and only have a single version, which is completely Open Source without using the GPL license, and WSO2 maintains this approach to the present day. This approach does leave the possibility that people will use the full enterprise-class product without paying. But in return it simplifies the model: when working with a community; accepting fixes; and encouraging true partnerships with customers - who become more willing to contribute to the codebase. It also creates a model where the success of the company is based on contented customers, not on license terms.

2 Changes in Open Source World

In the ten years since we started WSO2, there has been a large shift in the perception of Open Source. It is both better understood and less hyped. There is much more casual

[1] Industry standard valuations of Professional Services companies are roughly 2 – 3x revenue, whereas "Product" companies or companies with recurring revenue usually attract a valuation of 10x revenue.

use of 1Industry standard valuations of Professional Services companies are roughly 2 – 3x revenue, whereas "Product" companies or companies with recurring revenue usually attract a valuation of 10x revenue Open Source by companies, and governments in particular see it as a major differentiator when choosing software. However, in my experience, most commercial companies see it as just one of several factors but not the dominant one when choosing software. Ten years ago, it was seen as the "cheaper" alternative (e.g. MySQL vs Oracle). Today companies who understand Open Source well know that there are other more subtle reasons to choose Open Source: better knowledge of the code, more opportunity to influence the roadmap, more self-reliance. But the main benefit is that most Open Source projects fit well together into ecosystems and hence end up with simpler, cleaner solutions. The ability to contribute to an ecosystem is an important part of any new Open Source project or product: the ecosystems around Hadoop and Docker exemplify this.

3 Open Source Foundations

Many Open Source contributions are shepherded by foundations, such as Apache, Eclipse, OpenStack, and others. When we started WSO2, we modelled a lot of the company around practices from Apache and we also worked (and continue to work) closely with projects at Apache.

Foundations have three useful benefits to the community: independent ownership of the copyright; well-defined ground rules of operation (e.g. contribution, committership, IP rules, license, etc); and branding. However, the last few years have seen the rise of a different model: Github.

Although Github has no clear ground rules, and no enforcement or ownership of copyright, it has a strong brand and this has attracted millions of projects and forks.

4 The First Mover Advantage Is Over

In the last ten years, almost every single area has multiple open-source projects. Ten years ago, there was a considerable benefit to being the first well-known Open Source product in a market segment. Apache, MySQL, OpenOffice, and many others exemplify this. That time is over. Now, in order to succeed as an Open Source company you need to succeed just like any other company: by having a clearly visible competitive advantage independent of your software license.

Thanks

Thanks to Sanjiva Weerawarana, CEO and Founder of WSO2, and to all the team at WSO2 who have made it such a success.

How the Eclipse Community Works

Mike Milinkovich

Executive Director, Eclipse Foundation
Ottawa, Ontario, Canada
mike.milinkovich@eclipse.org

Abstract. Eclipse was the first of the open source foundations created as a consortium of corporations, starting a trend that has recently accelerated with the creation of corporate backed consortia for OpenStack, Cloud Foundry, OpenDaylight, and the like.

Mixing corporate and open source community interests in a single institution can be a recipe for conflict. But after 11 years of operation, the Eclipse Foundation remains remarkably functional and collegial. We have accomplished this by identifying a number of fundamental principles and sticking to them. This talk will describe those key principles and how we have implemented them within the Eclipse community. There will be lessons to be learned about how to balance the interests of many stakeholders, while remaining true to the core values of the broad open source community.

1 Foundations and Communities

There is a wide spectrum of open source project governance approaches, ranging from laissez faire, to Benevolent Dictator for Life, to community-supporting foundations. There is no shortage of opinion in the broader open source community about which approach is best. We do not view this as a conflict, but rather as a spectrum.

Projects and communities will have different needs and requirements during their lifetimes. In addition the other stakeholders in the ecosystem have their requirements. Something which is new and novel requires a very different governance structure than a project which has become widely adopted by industry.

2 Fundamentals Principles

The Apache Way first fully documented the key principles of openness, transparency and meritocracy, and those remain the bedrock of all well-governed open source project communities. Eclipse has added a couple of additional principles to that list such as vendor-neutrality, freedom of action, level-playing field for participants of all sizes, and community trademark control.

3 Balancing Stakeholders

It is a common assumption amongst the open source cognoscenti that the only acceptable approach to community and project governance is that the developers on the project

are responsible for all aspects of governance. The Eclipse model is a blended approach: projects are clearly self-governing meritocracies, but the governance of the Foundation itself pulls in a wider group of stakeholders.

These additional stakeholders include companies that have made a strategic commitment to Eclipse projects and technologies, companies which have adopted Eclipse technologies in their products, and (of course) the committers that work on Eclipse projects.

I will argue that this more diverse group of stakeholders has had a very positive affect on the overall governance.

4 Anti-Patterns

Eclipse was the first consortia-backed open source foundation, but in recent times it has been joined by quite a collection of others. Some of these have structural failings in their governance approaches. We'll take a looks at the anti-patterns in the consortia-led foundations.

Contents

Adoption, Use, and Impact

Intellectual Property and Legal Issues

OSS 2015 Ph.D. Contest

Open Source Software Engineering

An Empirical Study of the Relation Between Strong Change Coupling and Defects Using History and Social Metrics in the Apache Aries Project

Igor Scaliante Wiese[1(✉)], Rodrigo Takashi Kuroda[2], Reginaldo Re[1], Gustavo Ansaldi Oliva[3], and Marco Aurélio Gerosa[3]

[1] Departament of Computing, UTFPR – Universidade Tecnológica Federal Do Paraná/Campus Campo Mourão, Campo Mourão, Brazil
{igor,reginaldo}@utfpr.edu.br
[2] PPGI - UTFPR/Campus Cornélio Procópio, Cornélio Procópio, Brazil
rodrigokuroda@gmail.com
[3] Departament of Computer Science, IME/USP – University of Sao Paulo, Sao Paulo, Brazil
{goliva,gerosa}@ime.usp.br

Abstract. Change coupling is an implicit relationship observed when artifacts change together during software evolution. The literature leverages change coupling analysis for several purposes. For example, researchers discovered that change coupling is associated with software defects and reveals relationships between software artifacts that cannot be found by scanning code or documentation. In this paper, we empirically investigate the strongest change couplings from the Apache Aries project to characterize and identify their impact in software development. We used historical and social metrics collected from commits and issue reports to build classification models to identify strong change couplings. Historical metrics were used because change coupling is a phenomenon associated with recurrent co-changes found in the software history. In turn, social metrics were used because developers often interact with each other in issue trackers to accomplish the tasks. Our classification models showed high accuracy, with 70-99% F-measure and 88-99% AUC. Using the same set of metrics, we also predicted the number of future defects for the artifacts involved in strong change couplings. More specifically, we were able to predict 45.7% of defects where these strong change couplings reoccurred in the post-release. These findings suggest that developers and projects managers should detect and monitor strong change couplings, because they can be associated with defects and tend to happen again in the subsequent release.

1 Introduction

Some artifacts are changed together throughout software development. The concept of *change coupling* captures this implicit connection [1]. Some benefits of change coupling analysis were discussed by D'Ambros and colleagues [2]. For example, change couplings reveal relationships not present in the code or in the documentation. Other researchers showed that change couplings affect software quality [3,4]. Previous

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 3–12, 2015.
DOI: 10.1007/978-3-319-17837-0_1

studies discovered which artifacts changed together in the past by mining logs from version control systems, such as SVN and Git [1,5,6]. Zimmermann et al. [6], for example, implemented a tool to predict change propagation based on change coupling. The underlying assumption in their approach is that entities that changed together in the past are likely to change together in the future.

Differently from Zimmermann's work, our focus is to characterize and identify the impact of *strong change couplings*. Literature studies have provided some evidence that these couplings are associated with defects. D'ambros, Robbles & Lanza [4] mined historical data from three open source projects and showed that change couplings correlate with defects extracted from a bug repository. Cataldo et al. [7] reported that the effect of change coupling on fault proneness was complementary and significantly more relevant than the impact of structural coupling in two software projects from different companies. In another study, Cataldo and Nambiar [8] investigated the impact of geographic distribution and technical coupling on the quality of 189 global software development projects. By technical coupling, they mean overall measures of the extent to which artifacts of the system are connected. Their results indicated that the number of change couplings among architectural components were the most significant factor explaining the number of reported defects. Other factors they took into consideration include the number of structural coupling, process maturity, and the number of geographical sites.

In this sense, the main goal of this paper is twofold. First, we empirically investigate the relation between strong change couplings and the number of defects associated with them. Afterwards, we characterize strong change couplings using historical and social metrics.

We investigated the following research questions using data from the Apache Aries project:

- **RQ 1. Are strong change couplings related to defects?** We found that strong change coupling are associated with defects. In releases with more change couplings identified, more than 50% of them are associated with at least one defect. In releases with fewer change couplings identified, we found that at least ¾ of change coupling are associated with at least one defect. These values suggest that strong change couplings can be problematic for software projects.
- **RQ 2. Can historical and social metrics identify if a change coupling is strong?** We built models that identify strong change couplings with high accuracy (70-99% F-measure and 88-99% AUC). In addition, we applied the feature selection analysis to reduce the effort in building the prediction models and gain insights about which metrics are more important. We found that the length of a task discussion in the issue tracker, number of distinct committers, experience of committers, number of defect tasks associated with a change coupling, and age of change coupling were the best predictors.
- **RQ 3. Can we predict defects associated with strong change couplings?** We built a defect prediction model to help developers and managers to predict which strong change dependencies will have associated defects in the post-release. We correctly predicted 45.7% of the defects.

Results of our empirical study suggest that software engineers should detect and monitor strong change couplings, since they are associated with defects. Moreover, strong change couplings tend to happen again in the post-release to fix new defects.

This paper is structured as follows. In Section 2, we describe the methodology. In Section 3, we answer RQ1 and RQ2. Section 4 discusses the results of RQ3. Finally, conclusions and plans for future work are presented in Section 5.

2 Methodology

This section describes the methodology we followed to collect data and identify change couplings.

2.1 Identifying Strong Change Couplings

To identify the strong change couplings, we mined the change history of each release of the Apache Aries project. We considered just the changes submitted as a patch to an issue. If an issue had more than one associated commit, we grouped all commits in one single change transaction and, for each transaction, we employed a data mining technique called frequent itemset mining [1]. This technique was used in previous research [1,5,6] to uncover frequently occurring patterns (co-changed classes or methods) in a given set of transactions (change-sets/commits).

The frequency is typically measured by the metric of *support value*, which simply gives the number of transactions in which an itemset appears. In our study, the support value of an itemset consisting of files A and B corresponds to the number of issues in which they appeared together. The strength of a change coupling from A to B is determined by the ratio of co-changes (*support value*) and the number of times the artifact B changed. The artifact B in this example was the file that changed in more issues compared to artifact A. Based on these metrics, we used a quartile analysis to determine the "relevant" change couplings: all couplings with support higher than the third quartile were labeled as "strong". All other couplings were labeled as "weak."

2.2 Data Collection

In this paper, we collected data from the Apache Aries project, which delivers a set of pluggable Java OSGi components. We started the data collection extracting all issues from the Jira issue tracking system. For each issue, we collected its metadata and the associated source code changes from the version control repository. Since these two pieces of information were stored in different environments, we searched by words "defect, bug, fix" and an issue ID normally annotated by developers as "#"+issue number (e.g. #10). Using this query, we parse the commit messages and link each issue to their respective set of commits.

Table 1 summarizes the data collected from the Apache Aries project. It presents the release number, the number of issues, the number of change couplings, and the ratio of change couplings per issue for each release.

Table 1. Data collection summarization

Release	# of Issues	# of change couplings	Change couplings per issue	Duration of release (mm/dd/yy) - # months
0.1	194	4919	25.35	09/01/09 - 05/13/10 (8 months)
0.2	61	136	2.22	05/13/10 - 09/06/10 (4 months)
0.3	129	2465	19.10	09/06/10 - 02/21/11 (5 months)
0.4	85	469	5.51	02/21/11 - 11/08/11 (9 months)
1.0	62	300	4.83	11/08/11 - 10/12/12 (11 months)
1.1	25	48	1.92	01/29/13 - 01/23/14 (12 months)

We notice from the data above that Aries' releases have differences in the ratio of change couplings per issue report and in their duration. For example, release 0.2 lasted 4 months and involved fixing 61 issues and changing few files together. On the other hand, even though release 0.3 was one month longer, the number of fixed issues were higher (129) and many more files were changed together (2465) to fix these issues.

2.3 Classification Approach

We run the *random forest* technique to construct classifiers to identify strong change couplings. The random forest technique builds a large number of decision trees at training time using a random subset of all of the attributes. In our study, these attributes correspond to the historical and social metrics [9]. The technique performs a random split to ensure that all of the trees have a low correlation between them [9]. The random forest technique was already used in previous research [10].

Using an aggregation of votes from all trees, the random forest technique decides whether the final score is higher than a chosen threshold to determine if a specific change coupling will be deemed as strong or weak. To obtain the testing set and evaluate the performance of our classifiers, we used 10-fold cross-validation. Cross-validation splits the data into ten equal parts using nine parts for the training set and one part for the testing set.

We used two well-known metrics to evaluate our classifiers: F-measure and the Area Under the Curve (AUC). F-measure computes the harmonic mean of precision and recall for each class. AUC plots true positive rates against the false positive rates for various values of the chosen threshold used to determine whether a change coupling is classified as strong. The values of both metrics range from 0 to 1. Values close to 1 are desirable and indicate the best classifiers. We also analyzed the number of change couplings correctly predicted to further evaluate our classifiers.

3 Characterizing Strong and Weak Change Couplings

In this study, we conjecture that the set of strong change couplings are more relevant and consequently demands more attention from software developers In Section 3.1

(RQ1), we characterize strong change couplings by investigating if they are associated with software defects. In Section 3.2 (RQ2), we investigate whether historical and social metrics aids in the identification of strong couplings.

3.1 RQ1: Are Strong Change Couplings Related to Defects?

To answer this research question, we first counted the number of defect issues associated with strong change couplings in each release.

Table 2 depicts the number of instances labeled as strong change coupling with defects, the total of strong change couplings found, and the ratio of change couplings with defects. We found that the majority of the strong change couplings could be associated with at least one defect.

Table 2. Strong Change couplings (sCC) summary

Release	sCC with defects	Total of sCC	sCC with defect / total sCC (%)	By number of defects						
				1	2	3	4	5	6	7
0.1	719	1269	57.00%	531	151	20	15	2	0	0
0.2	9	9	100.00%	5	3	1	0	0	0	0
0.3	497	529	94.00%	128	276	67	19	3	5	1
0.4	63	82	77.00%	21	39	3	0	0	0	0
1.0	10	10	100.00%	0	6	4	0	0	0	0
1.1	3	4	75%%	1	2	0	0	0	0	0

To check just how correlated strong change couplings and defects are, Spearman's rank correlation coefficient (*rho*) was used. The Spearman correlation is a nonparametric measure of statistical dependence between two variables. The value returned by the Spearman correlation can range between +1 (positive correlation) to -1 (negative correlation). We calculated the correlation between the number of defects and the number of co-changes for each strong change coupling (considering all the releases in a whole).We found that strong change couplings are moderately correlated (*rho* 0.46, $p < 0.001$) with the number of defects. We noticed that releases 0.1, 0.3, and 0.4 have the majority of the strong change couplings that are correlated with defects. It is important to highlight that releases 0.1, 0.3 and 0.4 had more files changed and issues fixed compared to the releases 0.2, 1.0 and 1.1 (as shown in Table 1).

Considering the minor releases, we observed that the relation between strong change couplings and defects were higher, showing that at least 75% of the strong change couplings have at least one defect associated. Previous research also shows that, in general, change coupling is correlated with defects, both in open source [4] and industrial projects [3]. A possible reason for that is related to design issues that change couplings can be associated with. For example, some authors have associated these change couplings with the information hiding principle [11,12] described by Parnas [13]. The principle of information hiding indicates that two elements depending on the same internal class should be placed into the same module to hide the design decisions.

We found that strong change couplings are correlated with defects in the Aries project. We found positive correlation even when couplings happened in a small amount (fewer pairs of file co-changing).

3.2 RQ 2: Can Historical and Social Metrics Identify if a Change Coupling is Strong?

Previous research investigated the interplay between structural dependencies and change coupling [14,15]. They concluded that structural dependencies often could not explain or justify the emergence of change couplings. Thus, we do not yet have a clear idea of the nature of change couplings [7]. In this paper, instead of structural dependencies, we relied on historical and social metrics to build models and classify change couplings as "strong" or "weak." As we mentioned in Section 2.2, we distinguish between strong and weak change couplings based on a quartile analysis of their support value. Table 3 presents the number of change couplings per class and the values of F-measure and AUC.

Table 3. Prediction results to strong and weak change couplings using cross-validation 10-fold

Release	#strong	#weak	Total of Instances	F-measure Strong	F-measure Weak	AUC
0.1	1269	3650	4919	0.98	0.99	0.98
0.2	9	127	136	0.75	0.98	0.88
0.3	529	1936	2465	0.98	0.98	0.98
0.4	82	387	469	0.90	0.97	0.94
1.0	10	290	300	0.70	0.99	0.99

**The release 1.1 was used only as a test set

Table 4 presents the results when we train the models using data from a previous release to identify strong change couplings in the current release. The results show that for two releases we correctly predict more than 78% of all strong change couplings. However, the class imbalance problem in these cases affected the results for three releases (0.3, 1.0, and 1.1), since we got a few number of strong change couplings instances in the training set to perform the prediction in the following release.

To reduce the class imbalance problem, we grouped all previous releases to train the models and the next release to test. Table 5 presents the results of this new prediction model. Three (0.3, 1.0, and 1.1) out of four releases had better results. For example, in release 0.3 we noticed an improvement of 35.54% (20.79% to 56.33%). It is important to mention that in a practical scenario, a project may not have sufficient history to group the data. Furthermore, when classification models are used, the smallest the training set size, the lower the effort to build it.

Table 4. Prediction results to strong change couplings using a previous release to train and next release to test

Release Test	% of Correct predictions for strong CC	# of strong CC tested
0.2	100.00%	9 (9)
0.3	20.79%	110 (529)
0.4	78.04%	64 (82)
1.0	30.00%	3 (10)
1.1	0.00%	0 (4)

Table 5. Prediction results to strong change couplings using all previous releases to train and next release to test

Release Train	Release Test	% of Corrected strong CC predicted	# of strong CC tested
0.1 + 0.2	0.3	56.33%	298 (529)
0.1 + 0.2 + 0.3	0.4	76.82%	63 (82)
0.1 + 0.2 + 0.3 + 0.4	1.0	60.00%	6 (10)
0.1 + 0.2 + 0.3 + 0.4 + 1.0	1.1	100.00%	4 (4)

Our prediction model based on historical and social metrics accurately identified strong change couplings, with F-measure ranging from 70% to 98% and AUC ranging from 88 to 99%. As expected, grouping data from previous releases to predict subsequent change couplings improved the results. The percentage of correctly predicted strong change couplings range from 56% to 100%.

3.2.1 Which are the Best Metrics to Identify Strong Change Couplings?

To identify the best set of historical and social metrics to predict change couplings, we used the Correlation Feature Selection (CFS) algorithm. CFS evaluate subsets of features based on identified good feature subsets that contain features highly correlated with the classification, yet uncorrelated with each other. By using CFS, we wanted to reduce the training time, enhance generalization by reducing overfitting, and improving the model interpretability by finding the best metrics that predict strong change couplings. It is also important to find the best set of metrics to reduce the effort to collect the metrics and apply the models in practice.

Table 6 presents the number of selections made by the feature selection technique for each metric selected in at least one release. All metrics were sorted by their relevance. Wordiness (number 1 in the table) was the most important metric to identify strong change couplings. This metric was selected in four out of the five releases that were used to train the models.

In average, we concluded that 5 metrics by release were sufficient to identify strong change couplings. We deemed as relevant the metrics 1 to 6, since they were chosen in at least two different releases.

Table 6. Strong change couplings (CC) summary

	Software Metrics															
Release	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0.1	X															
0.2		X	X	X		X	X	X								X
0.3	X	X			X											
0.4	X	X	X	X	X	X					X	X	X	X	X	
1.0	X								X	X						

**blue columns are social metrics | white columns are historical metrics

1-wordiness, 2-devCommitSUM, 3-oexp, 4-oexp2, 5-taskDefect, 6-ageTotal, 7-devCommenters, 8-add, 9-clsMAX, 10-dgrSUM, 11-commits, 12-taskImprovement, 13-efvSizeMdn, 14-efvSizeMax, 15-MinorContributors, 16-devCommitAvg

*experience of committers is given by: dividing the total number of lines changed in the file in each release by the number of lines changed by each developer that changed the file in the same release. We got the maximum value of experience for each file involved in a change coupling.

**ageTotal is measured for each release and corresponds to the number of weeks in the period delimited by the first and the last commits in which the two files co-changed.

For the Aries Project, the best subset of metrics were composed by: length of discussion in terms of the number of words used (wordiness), number of distinct committers (devCommitSUM), experience of committers* (oexp, oexp2), number of defect tasks associated with a change coupling, and ageTotal**.

4 Application

4.1 RQ 3. Can we Predict Defects Associated with Strong Change Couplings?

To evaluate the applicability of our models, we wanted to check if pairs of files deemed as strongly change coupled in a release tend to co-change in the post-release. This gives us an idea of how many change couplings relationships "propagate" to the subsequent release.

Table 7. Prediction results to identify strong change couplings with defects

Release Train / Release Test	# of Strong CC with defects (next release)	% of correct predictions
0.1 / 0.2	40	60% (24)
0.2 / 0.3	7	100% (7)
0.3 / 0.4	30	20% (6)
0.4 / 1/0	4	0% (0)
1.0 / 1.1	0	-

Table 7 presents the defect prediction results for the strong change couplings identified in each release. For example, release 0.1 has 1270 strong change couplings (Table 2 – column total of strong change coupling). We found that 40 out of these

1270 couplings occur in at least one bug-fixing issue in the consecutive release. We labeled these 40 change coupling as "defective" and all the others strong change couplings as "clean." Performing the same machine learning analysis used in the previous section, we predicted whether a strong change coupling would have a defect.

We found that 81 strong change couplings happened again in the following release to fix defects and we correctly predicted 37 (45.67%) of them.

5 Conclusions

Our results show that strong change couplings are positively correlated with the number of defects. This corroborates previous results from the literature [4,7,8]. We noticed that by using the Random Forest machine-learning algorithm, it was possible to identify strong and weak change couplings for each release. In some cases, just the previous release was sufficient to train the models. In the cases where the previous releases had few strong couplings in the training set, we added all the previous history of strong and weak change couplings to improve the model accuracy. We were able to predict 45% of strong change couplings that happened again in the post-release to fix defects. These findings suggest that developers and projects managers should detect and monitor strong change couplings, since they propagate to future releases.

Potential threats to the validity can affect the results of our study. The first concern is the generalizability. In our analysis, we presented a single case study. However, based on this limited scope, our results might not generalize to other projects and domains. On the other hand, the choice of a single project allowed us to control more variables and better understand the data we collected. Another threat refers to the possible presence of tangled code changes [16] in the commits we mined.

As future work, we want to reevaluate our results on additional projects. We also want to go deeper and investigate the ways in which strong change couplings can influence code quality. This would serve as a basis for the development of new tools that would help managers monitor and track the damage caused by these couplings.

Acknowledgments. We thank Fundação Araucária, NAPSOL, NAWEB, FAPESP, and CNPQ (461101/2014-9) for the financial support. Igor receive grants from CAPES (BEX 2039-13-3). Gustavo receives individual grant from CAPES.

References

1. Gall, H., Hajek, K., Jazayeri, M.: Detection of logical coupling based on product release history. In: Proceedings of the International Conference on Software Maintenance, p. 190. IEEE Computer Society, Washington, DC, USA (1998)
2. D'Ambros, M., Lanza, M., Lungu, M.: Visualizing co-change information with the evolution radar. IEEE Trans. Software Eng. **35**, 720–735 (2009)

3. Kirbas, S., Sen, A., Caglayan, B., Bener, A., Mahmutogullari, R.: The effect of evolutionary coupling on software defects: an industrial case study on a legacy system. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 6:1–6:7. Torino (2014)
4. D'Ambros, M., Lanza, M., Robbes, R.: On the relationship between change coupling and software defects. WCRE 2009, pp. 135–144 (2009)
5. Ying, A.T.T., Murphy, G.C., Ng, R., Chu-Carroll, M.C.: Predicting source code changes by mining change history. IEEE Trans. Softw. Eng. **30**, 574–586 (2004)
6. Zimmermann, T., Zeller, A., Weissgerber, P., Diehl, S.: Mining version histories to guide software changes. IEEE Trans. Software Eng. **31**, 429–445 (2005)
7. Cataldo, M., Mockus, A., Roberts, J.A., Herbsleb, J.D.: Software dependencies, work dependencies, and their impact on failures. IEEE Trans. Software Eng. **35**, 864–878 (2009)
8. Cataldo, M., Nambiar, S.: The impact of geographic distribution and the nature of technical coupling on the quality of global software development projects. Journal of Software Maintenance and Evolution: Research and Practice (2010)
9. Breiman, L.: Random forests. Mach. Learn. **45**, 5–32 (2001)
10. McIntosh, S., Adams, B., Nagappan, M., Hassan, A.E.: Mining co-change information to understand when build changes are necessary. In: Proc. of the 30th Int'l Conf. on Software Maintenance and Evolution (ICSME), pp. 241–250 (2014)
11. Beck, F., Diehl, S.: On the congruence of modularity and code coupling. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 354–364. ACM, Szeged (2011)
12. Silva, L.L., Valente, M.T., de A. Maia, M.: Assessing modularity using co-change clusters. In: Proceedings of the 13th International Conference on Modularity, pp. 49–60. ACM, Lugano (2014)
13. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Commun. ACM **15**, 1053–1058 (1972)
14. Geipel, M.M., Schweitzer, F.: The link between dependency and cochange: empirical evidence. IEEE Trans. Software Eng. **38**, 1432–1444 (2012)
15. Oliva, G.A., Gerosa, M.A.: On the interplay between structural and logical dependencies in open-source software. Simpósio Brasileiro de Engenharia de Software, pp. 144–153 (2011)
16. Herzig K., Zeller, A.: The impact of tangled code changes. In: Proceedings of the 10th Working Conference on Mining Software Repositories, pp. 121–130. IEEE Press, San Francisco (2013)

Scaling and Internationalizing an Agile FOSS Project: Lessons Learned

Stephan Fellhofer(✉), Annemarie Harzl(✉), and Wolfgang Slany(✉)

Institute for Software Technology, Graz University of Technology,
Inffeldgasse 16b/II, 8010 Graz, Austria
stephan.fellhofer@gmail.com, annemarie.harzl@ist.tugraz.at,
wolfgang.slany@tugraz.at

Abstract. This paper describes problems that arose with the scaling and internationalization of the open source project Catrobat. The problems we faced were the lack of a centralized user management, insufficient scaling of our communication channels, and the necessity to adapt agile development techniques to remote collaboration. To solve the problems we decided to use a mix of open source tools (Git, IRC, LDAP) and commercial solutions (Jira, Confluence, GitHub) because we believe that this mix best fits our needs. Other projects can benefit from the lessons we learned during the reorganization of our knowledge base and communication tools, as infrastructure changes can be very labor-intensive and time-consuming.

Keywords: Agile development · Kanban · Distributed software development · Documentation management · Communication · Scaling · Internationalization

1 Introduction

Scaling and internationalizing a Free and Open Source Software (FOSS) project is not an easy task, at least not in our experience. Our project grew from five contributors in 2010 to over 130 in 2014, which lead to various organizational problems. In this paper we describe our problems, approaches we devised to solve them, and lessons learned along the way. We think that other FOSS projects can profit from our experiences.

In our project Catrobat we develop a visual programming language which has been inspired by Scratch [11] from the Lifelong-Kindergarten-Group at the MIT Media Lab. Our aim is to empower children and teenagers to easily create their own programs and to express themselves creatively using their smartphones. To reach this goal we develop Integrated Development Environment (IDE) and interpreter apps natively for several mobile platforms. Our app, Pocket Code, is available for Android on Google Play[1]. Versions for iOS, Windows Phone, and

[1] https://play.google.com/store/apps/details?id=org.catrobat.catroid

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 13–22, 2015.
DOI: 10.1007/978-3-319-17837-0_2

HTML5 capable browsers are currently in development. Catrobat was initiated by Wolfgang Slany and a team of students from Graz University of Technology, seeking development challenges beyond those seen in class and to practice what they have learned. This eagerness to apply software development principles taught in courses heavily influenced the basic structure of the project, with all its advantages and disadvantages.

On the positive side the usage of agile development methods such as Kanban [2,5] enables us to stay flexible and to easily adjust the scope of our project as we go. eXtreme Programming (XP) (especially pair programming [3]) facilitates knowledge transfer between developers. Test-Driven Development (TDD) [4] ensures that the code remains functional and testable while new developers join and former developers leave the project. On the negative side our contributors additionally have to learn about agile development methods, which steepens the learning curve.

A dedicated Usability and User Experience (UX) team applying the personas method [6] and other usability techniques helps us focus on the users. This is particularly important because, in contrast to most FOSS projects, Catrobat's developers (mostly university students) are not a subgroup of Catrobat's targeted users (mostly children and teenagers). The UX team helps to create an understanding of user needs in all developing teams.

Communication within our teams was initially mainly face-to-face, which is good for localized agile teams, but leaves many decisions undocumented, which is disadvantageous for a larger, distributed FOSS project. Discussions tend to get started over and over again, when nobody remembers why and based on what information a decision was made in the first place. Later we will elaborate on the communication problems, because they are at the core of our difficulties with scaling and internationalizing.

The increasing number of contributors from five in 2010 to over 130 in 2014, and the participation of international contributors in our project made organizational problems apparent. Documentation was not equally available for everyone, the entire current project status was not visible online and we lacked important communication channels. For example most of our contributors did not use Internet Relay Chat (IRC), which is often an integral infrastructure of FOSS projects and compulsory for the participation in Google Summer of Code (GSoC)[2], in which our project participates since 2011. To address these problems and to be able to integrate more and international contributors, we had to change our project infrastructure, the ways we communicate and some of our tools.

Section 2 gives a short overview of work related to the approaches we formulated. In Section 3 we will identify the problems in greater detail and describe our approaches to solve them. Section 4 contains the lessons we learned on our

[2] A global program from Google to support FOSS projects by sponsoring *students* and pairing them with *mentors* to develop given tasks over summer (https://developers.google.com/open-source/soc/)

way from a small, localized to a larger, more international project. Subsequently we discuss possibilities for future work and provide some concluding remarks.

2 Related Work

Our main goal was to enable and facilitate contributions from project members around the world. Various studies ([7,9,10,13,14]) highlight the importance of communication in FOSS projects or in projects which use agile development techniques.

We focused on optimizing the communication for FOSS and agile software development projects. Specifically, we tried to address some of the issues and approaches highlighted in the literature: Layman et. al [9] recommended among other things that *"when face-to-face, synchronous communication is infeasible, use an email listserv"* and *"use globally-available project management tools"*. Korkala et al. [7] suggested to *"enable and support direct communication between the developers"*. Some technologies and common practices used in FOSS development such as instant messaging, IRC, news postings, how-to guides, Frequently Asked Questions (FAQ), or wikis are listed in [12,13]. In [16] technologies such as versioning systems and TODO lists are mentioned. Difficulties for newcomers like *"selection of a suitable task"*, *"lack of up-to-date development documents"*, or *"no response from core developers for their doubts"* are mentioned in [15].

3 Optimizing Services for Distributed Participation

Our project grew faster than the supporting infrastructure, which led to organizational problems. For each part of our infrastructure we will first describe the initial situation, then problems which occurred over time and finally how we resolved them.

3.1 User Management

Initial Situation. In the beginning there was no consistent user management. Every piece of infrastructure (for example: instant messaging, source code repository) had its own built-in user management, and accounts were created manually on demand. Since some services were used through shared user accounts, there was no accountability.

Resulting Problems. The effort necessary for user management and maintenance increased tremendously with the growing number of contributors, because every user had to be created manually and every change had to be populated manually to all platforms. This resulted in missing and outdated account information. Rights management was not even a topic. Sometimes this lack of restrictions caused inexperienced contributors to inadvertently change or delete infrastructure. Shared accounts made it impossible to trace who made changes to project services. On the side of the contributors the account management was elaborate too, as for every service, contributors had to use different credentials.

Method of Resolution. Our goal was to simplify the management of user databases and to support the contributors by providing them with only one account for (almost) all our infrastructure. Most parts of our infrastructure have a built-in support for Lightweight Directory Access Protocol (LDAP), so LDAP was the most suitable solution to simplify our user management. We decided to use LDAP groups as well for various reasons:

- different experience levels need different rights
- different contributor groups need different resources and services
- no shared accounts, so there are clear responsibilities
- infrastructure administration should be left to experienced contributors

The goal was to design a user management which serves experts and beginners alike. Experts should have all the rights they need, while beginners are not overwhelmed by too many services and rights too soon.

Unfortunately, not all services support LDAP. For example, externally hosted services such as GitHub do not support foreign user directories and still need additional maintenance. Other services like Crowdin[3] support OAuth[4] as authentication method, but to take the user groups and corresponding rights needed for our project into account, the service's Application Programing Interface (API) or an additional configuration interface must be used.

3.2 Communication

Our project was and still is allowed to use a room at Graz University of Technology, where our local contributors can meet, discuss, and code. In the beginning all contributors fit in the room, and communication was mainly face-to-face. The reliance on face-to-face was very beneficial in the beginning but caused some communication and documentation problems later on as mentioned in Section 1.

Instant Messaging

Initial Situation. Other means of communication were and still are e-mail, mailing lists[5] as recommended in [9] and Instant Messaging (IM). In the beginning we used a Skype group chat for project discussions with all contributors.

Resulting Problems. Many contributors joined the Skype group chat but did not participate actively, because they preferred face-to-face communication, e-mail, or were overwhelmed by the number of messages that did not concern them. This massive number of messages was a direct result of the growing number of contributors.

Another problem with Skype was that group chats are invite-only, which made it difficult for aspiring contributors to join the discussion. They first had

[3] Tool to help non-developers to translate texts (https://crowdin.com/)
[4] http://oauth.net/
[5] https://groups.google.com/forum/?hl=en#!forum/catrobat

to find a project member to invite them. Yet another problem with our Skype group was the language used. Almost all messages were in German, because all of the initial team members spoke German.

In our attempt to open up our project and allow for internationalization, we created an IRC channel, which was open to everyone and where it was obligatory to use English.

This attempt failed, because project issues were, as a matter of habit, still discussed in Skype and hardly anyone used IRC. So theoretically we maintained an IRC channel, but interested contributors still got *"no response from core developers for their doubts and support request"* [15].

Method of Resolution. We decided to switch our whole synchronous communication to IRC and deleted the Skype group chat. The reasons for this decision were:

- one IM platform for all purposes
- anybody can join the channel he or she is interested in
- 'irrelevant' messages are reduced, because messages are posted only to the channels where they belong
- faster responses to questions from aspiring contributors due to increased online time of project members
- topic specific channels can be created and deleted easily, when needed

To make the communication with IRC more attractive for our contributors we provide them with an IRC bouncer which records all messages when the user is offline and replays them when the user goes online. In our attempt to minimize the amount of messages every contributor has to read, we decided to split the conversation into different channels. There exist separate channels for subteams, technical support teams (for example for our continuous integration server) and one channel for general information and announcements. Contributors may still miss important information, if they do not join all channels, but the risk of information overload should be diminished.

One disadvantage of IRC as communication platform is that the technology seems old-fashioned to our contributors and most of our contributors have never used IRC before. Another disadvantage is that it is more time consuming to configure IRC with the bouncer than it is to configure Skype. Contributors have to authenticate to freenode[6] and to the project bouncer with different credentials.

Today IRC is widely accepted by our community and the communication improved compared to Skype because contributors only have to join and read channels they are interested in and people are by now used to IRC.

3.3 Agile Development Management

Initial Situation. As already mentioned we were and still are allowed to use a room at the university for our project. There, we have whiteboards serving as

[6] IRC network where our project runs all its channels (https://freenode.net/)

Kanban boards, as well as story cards on paper at our disposal. Initially every subteam had its own Kanban board in the room, though this became impossible as the number of subprojects grew. Our project used and still uses GitHub[7] as source repository. Previously, we used the integrated issue tracker not just to track bugs and issues reported by users but also as a digital version of our local Kanban boards. Labels of issues were used to indicate which kind of issue it was (*bug, story, enhancement, ...*), the current working status (*to do, in development, done, ...*), the priority (*low, medium, high,* or *critical*) and which part it affected (*development environment* or *interpreter*). To enable children and teenagers to report bugs and issue requests without bothering with Github, we created a Google Group[8].

Resulting Problems. Our user stories and bug reports had to be synchronized between three different platforms, namely the local Kanban board, Github, and the Google Group. It is not hard to imagine that this had negative consequences: the local Kanban board was often outdated and the issue tracker did not cover stories, which were created locally on the Kanban board. The Github issue tracker did not support our Kanban board and lacked the functionality of hiding security relevant issues.

Method of Resolution. We decided to switch to an online distributed agile development environment. There exist various tools and we did a comparison of some, namely GitHub, Bugzilla[9], Redmine[10], and Jira[11]. We compared them based on the following criteria:

- they should support LDAP, virtual whiteboards, and different layers of visibility for security relevant issues
- the workflow should be customizable to our needs
- the tool should be expandable through add-ons
- means for reporting and analysis should be integrated
- a wiki system should be integrated because the old one needed to be replaced (for further details see Section 3.4)

We compared the basic versions of the different tools, without considering all the available add-ons, because add-ons can be more easily abandoned by their developers than complete tools, and we can not afford commercial add-ons. Table 1 shows the results of our comparison. Based on this comparison we decided to try Jira, because together with Confluence[12], it would satisfy all our criteria and both tools are free for FOSS projects.

[7] https://github.com/Catrobat
[8] https://groups.google.com/forum/?fromgroups=#!forum/pocketcode
[9] https://www.bugzilla.org/
[10] http://www.redmine.org/
[11] Planning and tracking tool for agile project management by Atlassian (https://www.atlassian.com/software/jira)
[12] Wiki system by Atlassian which is free for FOSS projects - https://www.atlassian.com/software/confluence

As a pilot test we used Jira during GSoC 2013. Every mentor/student pair received their own project with a simple Kanban board. Over this trial period we realized we had to customize the original workflow of issues to fit our developing principles (specifically review of newly created issues and code acceptance by experienced developers) and to integrate usability reviews into the workflow.

After the successful pilot test we decided to introduce Jira as a globally-available project management tool (recommended by [9]) and as replacement for the local whiteboards. This proved to be another important step towards becoming an international FOSS project.

Table 1. Comparison of bug tracker/project management software

	GitHub	Bugzilla	Redmine	Jira/Confluence
Free for FOSS	Yes	Yes	Yes	Yes
LDAP support	No	Yes	Yes	Yes
Restrict viewing rights	No	Yes	Yes	Yes
Workflow (customizable)	Manually with labels	Yes	Yes	Yes
Agile development support	3rd-party websites	Add-ons	Add-ons	Yes
Reporting	Basic graphs	Yes	No	Yes
Integrated wiki system	Yes	No	Yes	Yes
Expandable	Via API	Yes	Yes	Yes

3.4 Documentation Management

Initial Situation. Initially Google Docs, Dropbox, and a wiki system were used for distributing documents and information. The wiki system grew more or less naturally and the structure was seldomly revised. For a group of five people, which communicates mainly face-to-face, a less formal documentation management was useful, but once the number of contributors grew, we experienced severe problems.

Resulting Problems. Document owners left the project and the owner rights were not transferred. Documents were not updated, became outdated, and there was no general overview of documents and their content. File sharing tools like Google Drive and Dropbox were neither integrated in our wiki nor were they supporting our new user management with LDAP. These facts made it harder to share and find current information, documents, and their owners.

The wiki was not well maintained on all of its pages, and the organically grown structure could be an additional hurdle if one did not take the time to learn how to use the included tools. As discussed in [15], outdated development documents lead to difficulties for newcomers. Even some of our experienced project members tended to avoid using the wiki because of its partly outdated content and complex structure.

Method of Resolution. As explained in Section 3.3 we decided to use Jira as globally-available project management tool and Confluence as new knowledge management platform. The switch of systems provides us with an incentive to revise the structure and to eliminate or correct outdated information. This will make it easier to find up-to-date and useful information. The introduction of Confluence is still ongoing, because the information representation needs major rework.

Scacchi [13] introduced *Free and Open Source Software Development (FOSSD) informalisms*, which are easy to use and publicly accessible resources like: threaded discussion forums, group blogs, news postings, how-to guides, to-do lists, and FAQ. Most of these technologies are supported either by Jira or Confluence and we expect that they will support our contributors in their search for information.

4 Lessons Learned

4.1 Human Related

Introducing and switching to IRC as the new IM service was time consuming and resulted in resistance. We believe that the confusions described in Section 3.2, the out-dated and uncomfortable user interface of IRC clients, and our underestimation of the need for change management were mainly responsible for the long (and still ongoing) resistance. To reduce this opposition to change we used techniques described in [1,8] during the introduction of Jira. We involved more developers during the configuration period of Jira and the workflow adaptation. We communicated the benefits of the new system more clearly. We believe that this communication of benefits, training, involvement of GSoC mentors and optimizing the workflow led to a smooth change of our issue tracker and the introduction of Jira as a project management tool.

4.2 Technology Related

Centralized management of team member accounts and information about them should ideally be introduced from the very beginning as it tremendously simplifies the administration of contributors and saves a lot of time later. User rights management should be well structured and at the same time adjustable to future changes. Integrated services are preferable from a maintenance perspective, because they save time and effort related to organizational tasks. This time can then be spent on project goals. Not all external services, for example GitHub support LDAP, but if an API is available for that service, and user maintenance for this service consumes a lot of time, at least some parts should be automated by scripts.

5 Future Work

The transition to the newly deployed services is not yet finished for every team. Jira is already widely applied throughout the teams, but some teams still have

to make the transition. As stated in Section 3.4 Confluence is not yet deployed since the restructuring of the old system takes time and the focus is on easily accessible and maintainable data. Confluence will contain how-to guides, FAQ, blog-like news entries, and meeting notes to be more transparent and to provide new contributors with the most relevant and up-to-date information [13]. To support the transition [1,8] from the wiki system to Confluence we did a survey to detect main concerns and problems with the current wiki and are involving senior contributors in the content adaptation process.

6 Conclusion

As explained in Section 2 communication and documentation are essential parts of agile software development and even more important in distributed development [15]. Face-to-face communication is suitable at the beginning, but with the growth and internationalization of a project, good communication channels and project management tools have to be introduced. Every change of workflow or established tools is time consuming and needs proper change management to succeed. So before changing major aspects it should be well considered, if the changes are worth the effort. Appropriate user and rights management simplifies the administration of infrastructure and contributors. It saves time and supports contributors by giving them access to the project's services ideally with one account.

Acknowledgments. Thanks to all contributors of Catrobat[13] and also other supporting parties[14]. This work has been partially funded by the EC H2020Innovation Action No One Left Behind, http://www.no1leftbehind.eu/, Grant Agreement No. 645215.

References

1. Aladwani, A.M.: Change management strategies for successful erp implementation. Business Process management journal **7**(3), 266–275 (2001)
2. Anderson, D.J.: Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press (2010)
3. Andres, C., Beck, K.: Extreme Programming Explained: Embrace Change, 2nd Edition. Addison-Wesley Professional (2004)
4. Beck, K.: Test-Driven Development: By Example. Addison-Wesley Professional (2003)
5. Hiranabe, K.: Kanban applied to software development: from agile to lean (2008). http://www.infoq.com/articles/hiranabe-lean-agile-kanban, [Online; accessed 15-December-2014]
6. Hussain, Z., Lechner, M., Milchrahm, H., Shahzad, S., Slany, W., Umgeher, M., Vlk, T., Koeffel, C., Tscheligi, M., Wolkerstorfer, P.: Practical usability in xp software development processes. In: ACHI 2012, The Fifth International Conference on Advances in Computer-Human Interactions, pp. 208–217 (2012)

[13] http://developer.catrobat.org/credits
[14] http://developer.catrobat.org/special_thanks

7. Korkala, M., Abrahamsson, P.: Communication in distributed agile development: A case study. In: 33rd EUROMICRO Conference on Software Engineering and Advanced Applications, 2007, pp. 203–210. IEEE (2007)
8. Kotter, J.P., Schlesinger, L.A.: Choosing strategies for change. Harvard Business Review (1979)
9. Layman, L., Williams, L., Damian, D., Bures, H.: Essential communication practices for extreme programming in a global software development team. Information and Software Technology **48**(9), 781–794 (2006)
10. Poole, C.J.: Distributed product development using extreme programming. In: Eckstein, J., Baumeister, H. (eds.) XP 2004. LNCS, vol. 3092, pp. 60–67. Springer, Heidelberg (2004)
11. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al.: Scratch: programming for all. Communications of the ACM **52**(11), 60–67 (2009)
12. Scacchi, W.: Understanding the requirements for developing open source software systems. In: IEE Proceedings on Software, vol. 149, pp. 24–39. IET (2002)
13. Scacchi, W.: Collaboration practices and affordances in free/open source software development. In: Collaborative software engineering, pp. 307–327. Springer (2010)
14. Schümmer, T., Schümmer, J.: Support for distributed teams in extreme programming. In: Proceedings of eXtreme Programming and Flexible Processes Software Engineering - XP2000, pp. 355–377. Addison Wesley (2000)
15. Shibuya, B., Tamai, T.: Understanding the process of participating in open source communities. In: FLOSS 2009. ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, 2009, pp. 1–6. IEEE (2009)
16. Yamauchi, Y., Yokozawa, M., Shinohara, T., Ishida, T.: Collaboration with lean media: how open-source software succeeds. In: Proceedings of the 2000 ACM conference on Computer supported cooperative work, pp. 329–338. ACM (2000)

How Developers Acquire FLOSS Skills

Ann Barcomb, Michael Grottke, Jan-Philipp Stauffert,
Dirk Riehle(✉), and Sabrina Jahn

Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
{ann.barcomb,michael.grottke,jan-philipp.stauffert,
sabrina.jahn}@fau.de, dirk@riehle.org
http://www.fau.de

Abstract. With the increasing prominence of open collaboration as found in free/libre/open source software projects and other joint production communities, potential participants need to acquire skills. How these skills are learned has received little research attention. This article presents a large-scale survey (5,309 valid responses) in which users and developers of the beta release of a popular file download application were asked which learning styles were used to acquire technical and social skills. We find that the extent to which a person acquired the relevant skills through informal methods tends to be higher if the person is a free/libre/open source code contributor, while being a professional software developer does not have this effect. Additionally, younger participants proved more likely to make use of formal methods of learning. These insights will help individuals, commercial companies, educational institutions, governments and open collaborative projects decide how they promote learning.

Keywords: Competencies · Informal learning · Non-formal learning · Open source · Skills · Software developer

1 Introduction

Free/libre and open source software (FLOSS) is important to the economy, with many companies now relying on FLOSS components not only internally but also as the basis for their commercial offerings. Fostering FLOSS talent is critical for companies, which now support an estimated 50% of FLOSS development [1]. Governments have also become increasingly concerned not only with FLOSS adoption but with building capacity for FLOSS development as a means of promoting innovation. It is therefore in the interests of companies, governments and FLOSS communities to know which skills or competencies[1] are necessary for FLOSS development and how they can be trained.

[1] Some authors distinguish between competencies and skills while others do not. In this paper the terms are used interchangeably.

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 23–32, 2015.
DOI: 10.1007/978-3-319-17837-0_3

Contributing code to a FLOSS project requires both technical skills and social skills such as the ability to coordinate with others, to clearly articulate an argument [2], to give constructive feedback and to comply with social rules [3]. Basic information and communications technology (ICT) skills are described as including "the use of computers to retrieve, assess, store, produce, present and exchange information, and to communicate and participate in collaborative networks via the Internet" [4].

People can acquire skills through formal, non-formal and informal learning, often making use of multiple methods in acquiring a single skill [5]. Formal learning follows a course structure and results in certification, non-formal learning is structured but is not formally certified, while informal learning is neither structured nor formally recognized.

Although the skills necessary for FLOSS development have been identified, it is unclear which learning methods are being used to acquire them, especially by people who are not involved in FLOSS development. Our research shows that being a FLOSS code contributor and being a professional software developer have different effects on the learning methods used, and that age is an important factor in how FLOSS skills are acquired. We make use of an internet survey of users and developers of the beta release of a popular file download application to statistically validate the learning methods employed in the acquisition of FLOSS skills by group.

This paper makes three contributions. We confirm that the fact that someone is a FLOSS code contributor tends to make it more likely for the person to have used informal learning to acquire FLOSS skills, as discovered by Ghosh et al. [2]. We determine that being a FLOSS code contributor and being a professional software developer have different effects on the learning methods employed. Finally, we demonstrate that age affects the learning methods used.

2 Related Work

We identified three topics related to our research: the learning style preferences of individuals and groups, FLOSS and ICT skills acquisition, and comparisons between FLOSS code contributors and professional software developers.

Studies have examined the relationships between learning preferences and many other factors, such as hemisphericity ('right'- or 'left'-brain), social preferences, chronobiology and culture [6]. Age as an explanatory factor has recently attracted attention due to popular press claims that immersion in technology has created a generation of 'digital natives' with a uniquely self-directed and interactive learning style. Little support has been found for the premise in the literature [7]. A preference for informal learning was found in young students [8], but a preference for the informal techniques was observed irrespective of age in students aged from under 20 to over 30 [7]. In a study of Canadian adults, older people were found to make greater use of independent learning [9].

FLOSS code contributors strongly prefer informal methods such as reading source code and weakly prefer non-formal methods such as participating in workshops over formal study [2]. Contributors already possess many FLOSS skills

before joining the community [10], but younger cohorts do improve skills significantly within the community [11]. In contrast with these studies, our research looks at FLOSS code contributors in relation to others.

The learning styles being used to acquire technical skills, regardless of preference, are largely informal and non-formal. One of the largest surveys of skills acquisition indicates that the majority of Europeans acquired their ICT skills informally, through learning by doing, informal assistance from others, self-study materials, or through non-formal courses [4]. In a survey of Canadian adults, computer skills related to employment were among the skills most frequently acquired informally [9]. These studies focus on how skills are acquired, but they do not examine differences between groups. Our research shows that the same skills may be acquired differently by different groups of people.

FLOSS code contributors are often professional software developers or otherwise employed in the ICT sector [12]. They often already possess technical skills before joining a FLOSS project [13]. However, there are some demographic differences between FLOSS code contributors and professional software developers: FLOSS code contributors are more likely to be male [12] and come from North America or north-western Europe [14]. In terms of motivations, most FLOSS volunteers do not differ significantly from paid FLOSS code contributors [15], with most being motivated by need rather than altruism [16]. In FLOSS projects, a small percentage of people contribute most of the code [17], just as in other joint production communities. Top contributors in such projects participate in fundamentally different ways than others [18], and have different motivations [16]. Our research compares the effects of being a FLOSS code contributor and being a professional software developer and establishes that there are differences in learning methods used in the acquisition of FLOSS skills.

3 Theory Development and Hypotheses

We consider three methods of learning—formal, non-formal and informal—which differ on two key attributes: whether it is structured and if it includes certification. Formal learning "refers to the education received from a recognized education center that leads to a certification" [5]. "Non-formal learning is provided by any organised, structured and sustained educational activity... but typically does not lead to certification" [4]. "Informal learning is undertaken on one's own, either individually or collectively, without either externally imposed criteria or the presence of an institutionally authorized instructor" [9].

FLOSS code contributors expressed a strong preference for informal learning and a weaker preference for non-formal learning compared to the formal method of learning [2]. In this, they differ from respondents of another large survey (of government employees) [19]. Because the effectiveness of a learning method depends on how well it matches a person's learning style [6], we expect that FLOSS code contributors will make less use of formal learning than other respondents.

Hypothesis 1_a: Being a FLOSS code contributor makes it more likely that FLOSS skills have been acquired via informal learning methods.

Hypothesis 1_b: Being a FLOSS code contributor makes it more likely that FLOSS skills have been acquired via non-formal learning methods.

It has been demonstrated that there are differences between prolific contributors and ordinary contributors in open collaborative projects [16,18] but there is no indication that the average FLOSS code contributor differs from professional software developers. Indeed, many FLOSS code contributors work in the ICT sector [12]. Therefore we anticipate that being a professional software developer exhibits the same effects on the learning methods used for skills acquisition as being a FLOSS code contributor.

Hypothesis 2_a: Being a professional software developer makes it more likely that FLOSS skills have been acquired via informal learning methods.

Hypothesis 2_b: Being a professional software developer makes it more likely that FLOSS skills have been acquired via non-formal learning methods.

The majority of studies which examined the relationship between age and a preference for learning methods found no indication that age affects preferences [7,20].

While a preference for a particular method does not require that a person acquire a skill using that method, informal methods of acquiring technical skills are readily accessible and we expect that people will make use of their preferred methods of learning when the opportunity exists. We do not expect that age will affect the learning methods used in the acquisition of FLOSS skills.

Hypothesis 3_a: Age does not influence the extent to which FLOSS skills have been acquired via informal learning methods.

Hypothesis 3_b: Age does not influence the extent to which FLOSS skills have been acquired via non-formal learning methods.

4 Data Sources and Research Method

4.1 Data Sources

The primary data source for this paper is an online questionnaire conducted between December 2013 and January 2014 [21]. The survey was distributed to users of JDownloader 2 Beta via a link in the client interface. JDownloader[2] is an open source download management tool used by about 20 million people. A subset of users run the beta version.

The questionnaire was developed as a split survey with seven parts. In all parts, participants were asked common demographic questions and about their FLOSS participation. The distinct portion in each part related to the acquisition of FLOSS skills identified in prior work [2,3,22]. The questions connected with each of the skills examined (see Table 1 in Section 5) were ordered randomly and each appeared in two survey parts. Participants were randomly directed to one of the seven parts, resulting in a different number of responses for each question.

A second data source is the FLOSS 2013 survey [12] on demographics of FLOSS participants, which was conducted in late 2013.

[2] http://jdownloader.org/

4.2 Survey Reliability

An estimated 200,000 people use JDownloader 2 Beta, of which a total of 26,853 people started to answer one of the survey parts and 5,878 continued to the end.

We compared the completed responses to incomplete responses, and found that people who completed the survey were more likely to have engaged in software development and to have participated in FLOSS. Although the questionnaire stressed that the survey was intended for a broad audience, the focus on ICT may have discouraged some respondents.

We further eliminated responses where the participant failed to answer follow-up questions or where we suspected age misreporting because the response was outside the expected age range (born 1930–2000) of our population, leaving a total of 5,309 responses for the combined survey.

Internal reliability of the survey was demonstrated by computing Cronbach's alpha for the original versions of the skills questions against the control versions. All results were in the range of 0.6 to 0.9, which is considered acceptable.

4.3 Survey Representation

In our survey, we provided several ways for people to describe their FLOSS participation. We compared our respondents who selected from the five options which had close representations in FLOSS 2013 by gender, age and income. This comparison involved all FLOSS participants, not just FLOSS code contributors.

Using Pearson's Chi-squared test for gender we determined that there were differences, with our FLOSS participants being less likely to be female (1.4% compared to 11.1%).

To determine age in FLOSS 2013, needed for a t-test, we used the year of initial FLOSS participation and age at the time. Interval values with a range were adapted with random numbers from a uniform distribution within the range. Unbounded intervals were adapted as follows: "before 1960" was set to 1960, "10 or younger" was given a distribution from the set {8, 9, 10} and "55 or older" followed a distribution from {55, ..., 65}. A Welch two sample t-test showed with 95% confidence that our sample differed. Our sample was younger, as expected from the JDownloader population.

Income was expressed in intervals in both surveys. For a t-test, we converted the observations to values drawn from a uniform distribution within the interval limits and adjusted values to cover the same length of time. The results were consistent with a younger sample: our group had a lower average income.

4.4 Survey Design and Modeling Approach

Learning Style. Participants were asked to gauge their mastery of each skill shown in Table 1, by moving a slider between the extremes of "I am not skilled at all" and "I am very skilled." The maximum value corresponded with 10,000 but the numeric value was hidden from the participant. Participants who indicated some measure of skill were subsequently asked to evaluate to what extent various

methods were used to acquire the skill. Five options were presented: 'learning in school, university or apprenticeship' (formal); 'reading a book or online tutorial' (informal); 'observing other people perform the activity or the result of their work' (informal); 'participating in workshops or advanced training courses' (non-formal); and 'learning by doing' (informal). Learning styles were also displayed as unnumbered sliders with an effective range of 0–100 and a visible range of "nothing at all" to "all." Informal learning was favored by all participants for all skills, accounting for 62–80% of learning, while non-formal learning had a range of 9–16% and formal learning from 11–26%.

FLOSS Code Contributors. People who answered positively to the question "Have you ever participated in a FLOSS project?" and subsequently selected one or both of the participation options 'code contributions' and 'project founder' were categorized as FLOSS code contributors. The binary variable `FCC` was used to indicate if a respondent is a FLOSS code contributor. We observed that FLOSS code contributors differed from the rest of our sample by being younger (by 1.5 years) and less likely to be female (1.5% versus 3.4%).

Professional Developers. Professional software developers were classified by their selection of the answer "I work or worked in software development as part of my job" to the question "Have you worked in software development?" Based on this classification we created an indicator variable, `Prof`. It should be noted that all possible combination of values for the variables `FCC` and `Prof` occurred, giving us four different groups. The smallest group size was 196.

Age. Age was operationalized based on the year of birth, variable `YoB`, reported in the questionnaire by the respondents.

Technical Knowledge. We created a control variable, `TechK`, to describe an individual's technical knowledge. It contained the sum of the self-estimates of the mastery of technical skills (2, 5, 7, 11, 12 and 13) in Table 1.

Modeling Approach. For each skill, the vector $\boldsymbol{y} = (y_a, y_b, y_c)$ observed for a participant was assumed to follow a Dirichlet distribution with expectation $\boldsymbol{\pi} = (\pi_a, \pi_b, \pi_c)$ and precision ϕ, using the alternative parameterization proposed by Maier [23]. Here, y_a, y_b and y_c represent the observed relative learning acquired through informal, non-formal and formal learning styles, respectively, while π_a, π_b and π_c denote the corresponding expected values. Choosing formal learning as the reference category, the parameters were modeled to depend on the explanatory variables and the control variable as follows:

$$\ln\left(\frac{\pi_j}{\pi_c}\right) = \beta_{0j} + \beta_{1j} \cdot \texttt{FCC} + \beta_{2j} \cdot \texttt{Prof} + \beta_{3j} \cdot \texttt{YoB} + \beta_{4j} \cdot \texttt{TechK}, \quad j \in \{a, b\},$$

$$\ln \phi = \gamma_0 + \gamma_1 \cdot \texttt{FCC} + \gamma_2 \cdot \texttt{Prof} + \gamma_3 \cdot \texttt{YoB} + \gamma_4 \cdot \texttt{TechK}.$$

After estimating this model based on all observations available for a certain skill, we tested the null hypotheses $H_0(i_j) : \beta_{ij} \leq 0$ for $i \in \{1, 2\}$ and $j \in \{a, b\}$ using t-test statistics. If the related p value of such a test statistic was smaller than the chosen significance level α, then the null hypothesis could be rejected, indicating support for the respective alternative hypothesis, which was one of the Hypotheses 1_a–2_b formulated in Section 3. In contrast to this, the simple Hypotheses 3_a and 3_b formed the null hypotheses $H_0(3_j) : \beta_{3j} = 0$ with $j \in \{a, b\}$.

5 Results

When fitting the above Dirichlet regression model for each one of the 17 skills listed in Table 1, only those respondents could be taken into account who met the following conditions: they all had some mastery of the respective skill, they allocated a non-zero value to at least one learning method for acquiring this skill, they gave their year of birth, and they answered the questions necessary for determining their technical knowledge as well their membership in the FLOSS code contributor and professional software developer groups. Table 2 lists the sample sizes N available for the 17 models, as well as the p values p_{ij} obtained when testing the six null hypotheses $H_0(i_j)$, $i \in \{1, 2, 3\}$, $j \in \{a, b\}$. Results significant at a level α of 5% are shown in bold type.

Hypothesis 1_a was supported for skills 1–3, 5, 10–13, and 15. Descriptions of the skills can be found in Table 1. As expected, the fact that a person is a FLOSS code contributor tends to increase his/her use of informal learning methods in the acquisition of some FLOSS skills.

Table 1. Skills (skills shown in gray had control questions)

Skill #	Description
1	to evaluate the work of others
2	to work on own software module alone
3	to communicate with many different target groups
4	to understand English, especially technical discussion
5	to document code
6	to clearly articulate an argument
7	to understand different software architectures
8	to show respect for the work of others
9	to follow discussions on mailing lists
10	to communicate without offending others
11	to write code in a way that can be reused
12	basic/introductory programming skills
13	to acquaint yourself with code from others
14	to maintain contact with a community
15	to coordinate own work with the work of others
16	to change criticized behavior
17	to understand and work with people from different cultures

Table 2. Sample sizes and test results

Skill #	N	p_{1a}	p_{1b}	p_{2a}	p_{2b}	p_{3a}	p_{3b}
1	964	**0.0115**	0.5337	0.6664	0.0945	**0.0200**	**0.0000**
2	758	**0.0019**	0.1479	0.9986	0.9698	**0.0032**	**0.0000**
3	475	**0.0498**	0.8459	0.3033	0.4610	**0.0072**	**0.0122**
4	815	0.3313	0.8751	0.0847	0.0532	**0.0003**	**0.0000**
5	697	**0.0095**	0.2109	0.8809	0.9791	**0.0057**	**0.0000**
6	916	0.1645	0.0588	0.8712	0.4882	**0.0361**	**0.0000**
7	790	0.1832	0.6709	0.9999	1.0000	**0.0000**	**0.0000**
8	343	0.2783	0.2505	0.0640	0.4892	**0.0060**	**0.0043**
9	395	0.1483	0.4034	0.2757	0.1426	**0.0125**	**0.0020**
10	636	**0.0071**	0.2223	0.4565	0.1272	**0.0189**	**0.0000**
11	709	**0.0030**	0.0654	0.7705	0.6665	**0.0011**	**0.0000**
12	1046	**0.0000**	**0.0000**	1.0000	0.9956	**0.0028**	**0.0000**
13	591	**0.0133**	0.1295	0.2402	0.5800	0.1011	**0.0020**
14	350	0.0967	0.2230	0.1388	0.2587	0.1805	0.0641
15	751	**0.0144**	0.1769	0.6464	0.3472	**0.0005**	**0.0000**
16	527	0.3298	0.2965	0.4763	0.5724	**0.0227**	**0.0005**
17	390	0.3763	0.2838	0.3794	0.3056	**0.0158**	0.2393

Hypothesis 1_b was not supported, except for skill 12. Being a FLOSS code contributor does not make it more likely that non-formal learning methods have been used to acquire FLOSS skills.

Hypothesis 2_a and **Hypothesis 2_b** were not supported. The fact that a person is a professional software developer does not tend to increase his/her use of informal or non-formal learning methods in the acquisition of FLOSS skills.

Hypothesis 3_a was rejected except for skills 13 and 14. **Hypothesis 3_b** was rejected except for skills 14 and 17. For most FLOSS skills, age influences the use of informal and non-formal methods of learning. More specifically, our results indicated that for all skills where age has an effect, being older is associated with a higher likelihood of having acquired FLOSS skills informally and non-formally.

6 Discussion and Limitations

The learning methods used by FLOSS code contributors were expected, but the fact that not all skills showed this increased tendency toward informal learning suggests that future work is needed to determine why this variation exists.

There are important implications of the finding that in the acquisition of skills necessary for FLOSS development, being a professional software developer has a different effect from being a FLOSS code contributor. It has generally been assumed that the pool of potential FLOSS code contributors consists of all software developers, but our results suggest that there may be fundamental differences between professional software developers who contribute to FLOSS projects and those who do not. Future research should examine the extent of these differences, in order to determine if it is possible to encourage FLOSS

participation among software developers or if only a certain type of person—one who makes greater use of independent learning and exploration—is likely to become a FLOSS code contributor.

The effects of age on skills acquisition may reflect the availability of learning, rather than preferences. Previously, there were fewer formal options for acquiring FLOSS skills. This suggests it may not be futile to try to teach FLOSS skills, since they can be acquired through more formal methods. Future research could examine not only the methods by which skills were acquired, but the extent to which the skills were mastered. It should be noted that in our sample the age effect tended to offset the higher preference for informal learning among the FLOSS code contributors, because the FLOSS code contributors were younger.

Our sample was one of convenience optimized for response rate and is not representative of the general population, FLOSS code contributors, or software developers (see Section 4.3). Although this may limit the general applicability of our findings, the results are relevant for young adults, a group which is one of the most important targets for increasing FLOSS skills. Furthermore, as all our respondents came from the same population, our observations about the relative use of informal learning methods by different groups likely remain true.

7 Conclusions

In this paper, we presented a statistical analysis of a survey on the learning methods employed in the acquisition of FLOSS skills. We found that—unlike being a professional software developer—the fact that someone is a FLOSS code contributor tends to make it more likely for the person to have used informal learning methods to master a number of skills. Moreover, age strongly predicted differences in learning methods, with younger people proving more likely to make use of formal learning.

Our results provide some indication of how companies, FLOSS projects and governments can promote the acquisition of FLOSS skills, but also demonstrate the need for further research on how and to what extent FLOSS skills are learned.

Acknowledgments. The authors would like to thank Thomas Rechenmacher and JDownloader for promoting the survey and Nicole Kimmelmann for her contribution to the survey design.

References

1. Riehle, D., Riemer, P., Kolassa, C., Schmidt, M.: Paid vs. volunteer work in open source. In: Proc. 47th Hawaii Int. Conf. System Sciences. pp. 3286–3295 (2014)
2. Ghosh, R.A., Glott, R., Krieger, B., Robles, G.: Free/libre and open source software: Survey and study. Tech. rep., International Institute of Infonomics, University of Maastricht (2002). http://flosspols.org/deliverables.php
3. Kimmelmann, N.: Career in open source? Relevant competencies for successful open source developers/Karriere in Open Source? Relevante Kompetenzen für erfolgreiche Open Source Entwickler. it-Information Technology **55**(5), 204–212 (2013)

4. Ala-Mutka, K.: Review of learning in ICT-enabled networks and communities. Tech. Rep. 24061, Institute for Prospective Technological Studies (2009)
5. Galanis, N., Mayol, E., Alier, M., Garcia-Peñalvo, F.J.: A social framework for supporting, evaluating and validating informal learning. In: Proc. 2nd Int. Conf. on Technological Ecosystems for Enhancing Multiculturality, pp. 589–594 (2014)
6. Dunn, R., Beaudry, J.S., Klavas, A.: Survey of research on learning styles. California Journal of Science Education **2**(2), 75–98 (2002)
7. Lai, K.W., Hong, K.S.: Technology use and learning characteristics of students in higher education: Do generational differences exist? British Journal of Educational Technology (2014)
8. Hong, K.S., Aziz, N.A.: Technology use and digital learning characteristics among malaysian undergraduates. Sains Humanika **2**(1), 117–124 (2014)
9. Livingstone, D.W.: Exploring the icebergs of adult learning: findings of the first Canadian survey of informal learning practices (1999)
10. Ghosh, R., Glott, R.: Flosspols: skills survey interim report: 32. MERIT, University of Maastricht, Maastricht (2005)
11. Glott, R., Meiszner, A., Sowe, S.K.: FLOSSCom phase 1 report: analysis of the informal learning environment of FLOSS communities (2007). http://kn.open.ac.uk/public/getfile.cfm?documentfileid=12042
12. Arjona-Reina, L., Robles, G., Dueas, S.: The FLOSS2013 free/libre/open source survey (2014). http://floss2013.libresoft.es
13. Fang, Y., Neufeld, D.: Understanding sustained participation in open source software projects. Journal of Management Information Systems **25**(4), 9–50 (2009)
14. Takhteyev, Y., Hilts, A.: Investigating the geography of open source software through GitHub (2010). http://www.takhteyev.org/papers/Takhteyev-Hilts-2010.pdf
15. Lakhani, K.R., Wolf, R.G.: Why hackers do what they do: understanding motivation and effort in free/open source software projects. In: Feller, J., Fitzgerald, B., Hissam, S., Lakhani, K.R. (eds.) Perspectives on Free and Open Source Software, pp. 3–22 (2005)
16. Shah, S.K.: Motivation, governance, and the viability of hybrid forms in open source software development. Management Science **52**(7), 1000–1014 (2006)
17. Kagdi, H., Hammad, M., Maletic, J.I.: Who can help me with this source code change? In: Proc. IEEE Int. Conf. Software Maintenance. pp. 157–166 (2008)
18. Panciera, K., Halfaker, A., Terveen, L.: Wikipedians are born, not made: a study of power editors on wikipedia. In: Proc. ACM 2009 Int. Conf. Supporting Group Work, pp. 51–60 (2009)
19. Grundmann, S.T.: Making the Right Connections: Targeting the Best Competencies for Training. DIANE Publishing (2011)
20. Byrne, P., Lyons, G.: The effect of student attributes on success in programming. ACM SIGCSE Bulletin. **33**, 49–52 (2001)
21. Jahn, S.: Teaching open source competency (2014). http://osr.cs.fau.de/2014/04/02/final-thesis-teaching-open-source-competency/, bachelor thesis
22. Kimmelmann, N.: Private communication (2013)
23. Maier, M.: DirichletReg: dirichlet regression for compositional data in R. Tech. Rep. 125, Vienna University of Economics and Business (2014)

Communication and Collaboration

Implicit Coordination: A Case Study of the Rails OSS Project

Kelly Blincoe(✉) and Daniela Damian

University of Victoria, Victoria, Canada
kblincoe@acm.org, danielad@uvic.ca

Abstract. Previous studies on coordination in OSS projects have studied explicit communication. Research has theorized on the existence of coordination without direct communication or *implicit coordination* in OSS projects, suggesting that it contributes to their success. However, due to the intangible nature of implicit coordination, no studies have confirmed these theories. We describe how implicit coordination can now be measured in modern collaborative development environments. Through a case study of a popular OSS GitHub-hosted project, we report on how and why features that support implicit coordination are used.

1 Introduction

There are many large Open Source Software (OSS) development projects that succeed despite spanning geographic, organizational and social boundaries. Such boundaries normally create coordination barriers and make coordination more expensive [1]. Previous research hinted at the promise of implicit coordination, defined as coordination *"reached without discursive communication, shared plans or even previous commitment among the actors"* [2], in reducing coordination overhead on OSS projects [2–4]. Bolici et al. [2] identified cases where dependencies between tasks existed with no evidence of explicit communication and theorized that implicit coordination was used to fulfill these dependencies. However, since implicit coordination occurs without explicit communication, it is difficult to examine, and no studies have confirmed these theories.

Modern software development tools provide unprecedented support for implicit coordination in OSS projects. Developers can understand relevant tasks without interrupting the developers assigned to those tasks for explicit communication. For example, details of a dependent task can be reviewed to gain awareness about the task. Developers can document all task related decisions within task reports and insert comments directly into the source code, all of which can be easily reviewed by others. Additionally, tools like GitHub provide notifications of project activity to help developers stay aware. This transparency supports implicit coordination and makes collaboration easier [5].

We describe how modern development environments enable implicit coordination. We report on a case study of a popular OSS project hosted on GitHub. GitHub is a code hosting service and collaboration environment. Its transparency

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 35–44, 2015.
DOI: 10.1007/978-3-319-17837-0_4

[5] and built-in social features enable implicit coordination. In a mixed-method research approach, we surveyed 986 developers, interviewed 14 developers and conducted a repository analysis. We find that two GitHub features that support implicit coordination —issue subscriptions and following relationships —are used frequently and report on how and why these features are used.

2 Implicit Coordination

Previous studies on coordination in OSS projects [6,7] studied explicit coordination mechanisms likes emails and bug report comments. Coordination that occurs without explicit communication is known as implicit coordination, which consists of consequential communication and feedthrough [8]. Consequential communication is watching a developer complete their tasks to learn about their activities. Feedthrough is obtaining information about tasks by examining changes to artifacts and is an example of stigmergy. Stigmergy is a concept from biology that states *"work done by one agent provides a stimulus that entices other agents to continue the job"* [4]. Stigmergy occurs when enough information is contained within an artifact or a task report to enable a new developer to complete an ongoing task or start a new dependent task without explicit coordination. This occurs frequently on OSS projects [4]. Even independent tasks build on the work of others [9], so stigmergy plays a key role in OSS development.

Awareness is *"an understanding of the activities of others, which provides a context for your own activity"* [10]. Developers need to be aware of tasks (and associated artifact changes) and people [11]. A lack of awareness can result in coordination breakdowns [12,13]. Previous research found that, on OSS projects, developers obtain awareness through explicit communication [8], but it can also be achieved through implicit coordination. Modern software development tools make development work more visible and transparent [5] providing awareness and potential for implicit coordination. Therefore, the study we present here investigated awareness and implicit coordination and was guided our research question:

How are the features that enable implicit coordination being used on modern software development environments?

We describe features of modern software development environments that support awareness of tasks and people and enable implicit coordination in OSS projects. We then describe our study to address this research question.

3 Implicit Coordination Enabled by Modern Development Environments

While explicit coordination is characterized by communication, implicit coordination is achieved by obtaining awareness of the information needed to complete a task without communication. Modern software development environments have introduced features that enable implicit coordination. We describe how modern

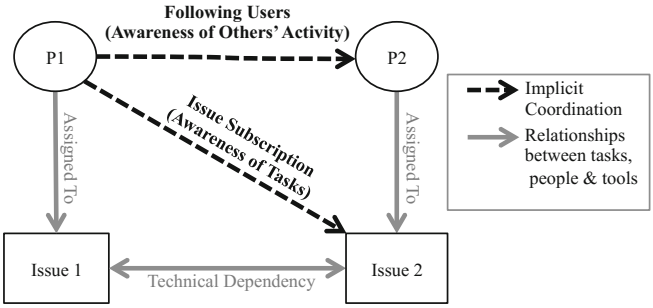

Fig. 1. Awareness Mechanisms for Implicit Coordination in GitHub

development environments support implicit coordination by enabling awareness of tasks and people. We specifically highlight the GitHub features that support implicit coordination and illustrate them in Fig. 1. Issues are GitHub's representation of tasks, so we refer to issues and tasks synonymously. In Fig. 1, a coordination need exists between developers P1 and P2 due to the technical dependency that exists between their tasks.

1) Awareness of Tasks

In a previous study, we found that developers prefer to review task details to understand a task rather than interrupting the task assignee to ask about the task [14]. Reviewing details of related tasks and the changes made to the source code as a result of those tasks can help developers gain an understanding of dependent tasks. Task details can be obtained from the task report in the team's issue or bug tracker. Reviewing artifacts in the source code management system can make developers aware of code changes, and comments left in the code may provide insight into why the changes were made.

Developers can obtain awareness of tasks by ***subscribing to feeds***. Feeds broadcast project-related or user events such as incoming issues or code changes. Developers use feeds to track work and get information [15].

Support in GitHub: GitHub supports awareness of tasks through its issue subscription feature. When users are subscribed to issues, they receive notifications of the activity occurring around that issue on their GitHub dashboard and, if configured, via email. Users are automatically subscribed to issues when they comment on or are tagged in a comment on that issue. Users can unsubscribe if desired. In Fig. 1, developer P1 becomes aware of task 2 through issue subscription.

Other Development Environments/Tools: SourceForge, another web-based code management tool, allows users to subscribe to issues through an RSS feed. Other issue trackers, like Jira and Bugzilla also allow users to obtain notifications of the activity occurring around issues. Jira has an issue subscription feature similar to the one provided in GitHub. The cc feature in Bugzilla works similarly, allowing developers to add themselves to a change request's cc list to receive notifications about that change request.

2) Awareness of Others' Activity

Developers can gain an understanding of others' activity by ***reviewing their work***. This was previously accomplished by monitoring version control check-in logs [8] and has been made easier through feeds that broadcast user activity. Many software development tools allow users to *follow* users. When someone follows a user, they receive notifications about that user's activity in their feeds.

Support in GitHub: Users can follow others by clicking on the follow button in a user's profile. The follower will then receive notifications about that user's activity across all GitHub projects. In Fig. 1, developer P1 is following developer P2 and, therefore, is aware of the activity of P2.

Other Development Environments/Tools: SourceForge enables following of other users through RSS feeds. In Bugzilla and Jira, you can search for issues or bugs a user is participating on, but feeds of a user's activity is not available.

Developers can gain a further understanding of others' activity by ***reviewing information shared through social media***. Software developers have adopted social media tools like *wikis, blogs* and *microblogs*. Studies have found that sharing information through these forums allows easy access to knowledge and can serve as a coordination method [16].

Support in GitHub: GitHub does not allow developers to share information about their activity in wikis, blogs or microblogs. External tools are often used in conjunction with GitHub such as Twitter [16] for this functionality.

Other Development Environments/Tools: No currently adopted development tools offer integrated blogging or microblogging.

4 Case Study

We examined Ruby on Rails, a popular project hosted on GitHub, often referred to simply as Rails. Rails is an open source web application framework written in the ruby programming language. Many companies and other OSS projects use the framework for creating web applications. Rails, therefore, attracts many contributors looking to fix bugs affecting their own products or add new features that are useful for them. We choose to study Rails since it has a large group of code contributors, resulting in frequent coordination needs.

To answer our research question, we collected and analyzed both qualitative and quantitative data through a survey, developer interviews, and a statistical analysis of Rails project repository data. We applied a grounded theory approach on the survey and interview data [17]. For the project repository analysis on Rails, we obtained data from GHTorrent [18], which provides a mirror of the GitHub API data. GHTorrent obtains its data by monitoring and recording GitHub events as they occur. We selected the project with the most code contributors in the GHTorrent 2014-04-02 dataset —Rails. We analyzed the most recent release available in that dataset, 4.0, which was developed from January 20, 2012 to June 25, 2013. Using GHTorrent, we obtained issue subscription information for all issues and following relationships for all users. We included

data from the main branch and any associated forks as recommended in [19]. We included all 2,437 issues that were closed during that release. There were 7,935 users who contributed through commits, comments, or issues. We refer to users and contributors interchangeably since we are studying only Rails contributors.

4.1 Methods

To answer our research question, we analyzed data from an online survey, interviews with 14 contributors, and the project repository.

Survey instrument. We sent an online survey to all 7,492 GitHub users with valid email addresses that participated on Rails during the release of interest. Any user who performed one of the following actions was identified as a participant: committed code; created an issue; submitted a pull request; commented on a commit, issue or pull request; closed, merged or reopened an issue or pull request; or subscribed to an issue.

We asked survey participants if, how and why they used the features that support implicit coordination in GitHub, subscribing to issues and following users. The questions in the survey were both multiple-choice and open-ended. Our survey is available at http://web.uvic.ca/~kblincoe/survey.pdf. We used unbalanced (skewed towards the positive) rating scale questions since we expected mostly positive answers and wanted to measure the degree of the responses [20]. We received 986 responses (12.4% response rate). We used standard qualitative coding techniques [17] to categorize responses and identify themes.

Interview instrument. To gain a better understanding of how implicit coordination takes place, we interviewed 14 of the survey respondents who volunteered for interviews. We randomly selected participants from the 19.2% of respondents who identified themselves as currently active participants of the Rails project. Interviews were semi-structured and lasted 30 minutes on average. The interviews were focused on coordination with questions like 'How do you know what others are working on?' and 'Are there ways you stay aware of project activity and avoid duplicate work or conflicts in your own work without explicitly communicating with other teammates?' Similar to the survey data, we used a grounded theory approach in our analysis of the interview transcripts [17].

Repository analysis. We examined how the features that support implicit coordination are used in GitHub by examining their frequency of use.

4.2 Results

1) Awareness of Tasks: Issue Subscription

Issue subscriptions are used extensively on Rails. 79.7% of issues have at least one subscriber. Of the issues with subscriptions, many (35.6%) have subscribers who have not commented or been tagged within a comment. For issues with subscribers, there is an average of 4.1 subscribers per issue (median is 3).

Many contributors subscribe to issues. Through repository analysis, we found that 48.1% of contributors are subscribed to at least one issue. Users who subscribed to issues were subscribed to 3.9 project issues on average (median is 1).

Our interviewees talked about how the notifications make it easier to stay aware of what is going on. While you can see what others are working on by looking through the code to identify changes, issue notifications are much more efficient.

"Trying to look at the actual code, like the todos or comments in the code or that people have open branches on that, that ends up being really difficult. You can spend a long time looking and not figure it out." [P10]

Many contributors subscribe to issues they are not participating on. Nearly half (49.4%) of our survey respondents said that they subscribe to issues that they are not actively participating on (where active participation considers reporting the issue, developing code or commenting on the issue). This is important because it implies that they are using these notifications for awareness of tasks that they are not working on (like developer P1 in Fig. 1 becomes aware of Issue 2 by subscribing to that issue even though she is not working on that issue).

"[Issue notifications] are the best way to passively stay up to date with things that are going on." [P11]

In fact, many of our survey respondents (65.9%) stated that they use the information obtained from issue notifications differently than when they are actively participating on an issue. When actively participating on an issue, they "use the notifications for more direct actions rather than just a general feeling of what's going on." They treat the notifications as a 'to-do' list. When they are not actively participating on the issues, they use the notifications more passively. They gave several reasons why they would subscribe to notifications for issues they are not participating on:

- *Dependencies.* 31% of respondents explicitly state that they subscribe to issues because of dependencies in their code or because they are experiencing the same issue. One respondent stated, "Something I'm working on could be depending on the outcome of the resolution." Another respondent said, "I ran into the same issue ... when they fix it, I will update to [the new version]."
- *Issue Status.* 31% of respondents simply reported that they subscribe to know when the issue is resolved without giving a more detailed reason.
- *Project Status.* 22% subscribe to maintain an overall awareness of project activity. "For some projects that I'm not actually interested in contributing actively, [subscribing to issues] is the closest you can get to a newsletter to get to know what new features are getting in and other general stuff."
- *General Interest / Education.* 9% subscribe because of general interest in the issue or to learn from the discussion or solution. One respondent said, "I like to see the changes and comments made by others to learn more."
- *Awareness for Future.* 7% subscribe to obtain knowledge for future development. One respondent said the notifications around certain issues "may help me make changes in the area in the future more easily". Other respondents noted they follow certain issues so they can offer help if needed.

One of the most common reasons for receiving notifications (31%) is to obtain information on dependencies. This is a form of implicit coordination. The respondents who receive notifications to provide awareness for the future (7%) are also

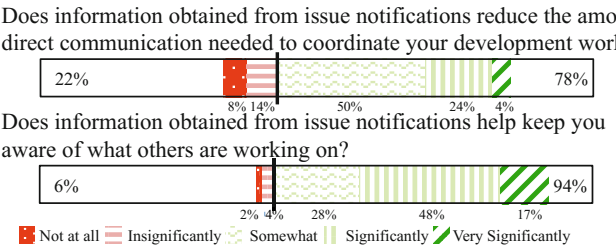

Fig. 2. Survey Responses around Issue Notifications

implicitly coordinating. They obtain awareness about code changes so they are familiar with the code and the design decisions to be able to contribute in the future more efficiently. Only a small number of our survey respondents subscribe to issues out of general interest (9%). Our interviewees only subscribe to issues that affect them directly.

> *"I used to try to follow things that I was just interested in to know what was going on with them, and I found that it was totally irrelevant. If I'm not actively using a repository, I really don't need the notifications."* [P5]

Subscribing to issues helps reduce communication and increase awareness. Fig. 2 shows that 78% of survey respondents found that the information obtained from issue notifications can reduce the amount of direct communication and 94% said it can increase awareness of what others are working on. One respondent said, "I can stay up-to-date on others thinking and inputs on a specific topic without needing to talk to them." Another respondent focused on the awareness he obtains from the information contained in issue notifications saying, "through their comments and patches I can see what others are working on and what their progress is." Our interviewees talked about how the notifications help reduce direct communication around issue status.

> *"If I am waiting for a fix on something, then the notifications will help me stay up-to-date . . . I don't have to contact the developer for the status."* [P5]

Interviewees also noted that communication is often limited to only when differences of opinion occur.

> *"I might see that someone is working on a particular issue and . . . take a look at the code that has actually been implemented and see if it is along the same lines as what I was thinking, kind of a preview. So if somebody is off on the wrong track or going down a code path that I don't think is actually going to fix the bug or implement that feature then I might send them an email or hop in a chat."* [P9]

The notifications also help keep developers aware of dependencies or work that has been done on related issues, according to our interviewees.

> *"If we have multiple pull requests that rely on each other. . . just seeing comments come across and the pull requests come in, helps me know where that is in the process."* [P6]

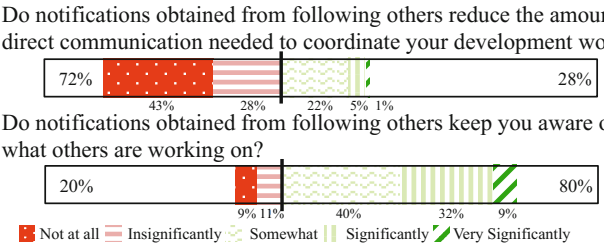

Fig. 3. Survey Responses around Following

2) Awareness of Others' Activity: Following

Many users follow other users. 66.7% of survey respondents say they follow other GitHub users. This is consistent with our repository analysis, which found that 64.3% of all Rails contributors follow other GitHub users. The reasons for following others are quite varied:

- *Track activity.* 41.8% follow others to see what they are up to and track their activity. "As a means of seeing what they are working on. Their contributions are working towards solutions in my general problem space." This reason is the most suggestive of implicit coordination, unlike the ones that follow.
- *Learn about new projects.* 23.8% follow others to see what projects they contribute to or star. "I have discovered a lot of projects from looking at what users I am following are looking at."
- *Social.* Many respondents (21.2%) said they follow others for social reasons. "That's basically the 'friend' feature on GitHub. I just follow others I know."
- *Useless.* Some survey respondents (6.8%) stated that the following feature is useless. "[It] is not really a useful feature because it adds too much noise."
- *Education.* 3.6% said they follow others to learn. "I follow experts in certain code bases to educate myself by reviewing their check-ins and comments."
- *Bookmark users.* 2.8% said they follow others to 'bookmark' interesting GitHub users since it "makes it easier to find their profiles in future."

Users often follow other contributors on their projects. 46.1% of Rails contributors are following other Rails contributors. Those who are following other contributors are following 7.2 project contributors on average (median is 3).

Following other users does not reduce communication but does increase awareness. Fig. 3 shows 72% of survey respondents found that the notifications obtained from following others does not reduce the amount of direct communication. However, 80% found the notifications did increase awareness of what others are working on. One respondent said, "I can see other people's activity and what projects they are actively working on." Our interviewees noted that the notifications from following others is too high-level to be useful for coordination.

> *"I just see the repositories they create, the repositories they fork, what they star and stuff like that. . . . It is just a general overview of what they are doing, and if I want to know anything in detail, I have to still ask them or go to them or just look at the code to see what's going on."* [P2]

In addition, since following others results in notifications related to their activity on all repositories, the notifications can contain a lot of noise.

"When I follow a person, they work on an assortment of repositories, most of which have nothing to do with me probably. I follow very few people." [P5]

5 Discussion

The findings from our study of Rails, a prominent OSS project hosted in GitHub, indicate that both issue subscription and following other users are widely adopted by OSS users. There are various reasons why users choose to use these features. The main reason for subscribing to issues is to obtain information on dependencies, a form of implicit coordination. Survey respondents believed that subscribing to issues reduced their direct communication.

However, many of our interviewees and survey respondents indicated that notifications from following others introduced too much noise and, therefore, were not useful. Additional surveys or interviews could shed some light on what notifications are most useful for implicit coordination so the notifications can be minimized to only the most relevant information. Important research questions include: What is the most useful information for inclusion in notifications of other's activity? Are users influenced by the actions of the users they follow?

The effect of explicit coordination has been studied by quantitatively assessing how the coordination structure of a team aligns with the teams' coordination needs. Conway's Law [21] was the first to introduce the idea of such an alignment. Now that implicit coordination can be measured through features like issue subscription and following, future research can study the impact of implicit coordination. If implicit coordination improves productivity and quality, managers can encourage implicit coordination to reduce coordination overhead. Further, tools that provide coordination recommendations to developers could focus on less expensive, implicit means of coordination. While bringing new knowledge about indirect collaboration in modern, open development environments, this study is only the beginning of our exploration into indirect coordination. We investigated only one project in detail; thus, it suffers from some threats to validity regarding generalizability. Our survey respondents and interviewees were all Rails contributors and were self-selected. We reached saturation in our results, but they may not generalize to other GitHub projects. However, many of our interviewees were active contributors to many GitHub projects and their responses drew on their experience across multiple projects. Additional studies can continue this investigation on other GitHub projects. Further, this investigation can be continued through future studies of implicit coordination in other modern development environments like SourceForge, Jazz or Bitbucket.

References

1. Herbsleb, J.D.: Global software engineering: The future of socio-technical coordination. In: FSE 2007, pp. 188–198. IEEE Computer Society (2007)
2. Bolici, F., Howison, J., Crowston, K.: Coordination without discussion? socio-technical congruence and stigmergy in free and open source software projects. In: STC 2009 (2009)
3. Elliot, M.: Stigmergic collaboration: The evolution of group work. m/c journal **9** (2006)
4. Heylighen, F.: Why is open access development so successful? stigmergic organization and the economics of information. arXiv preprint cs/0612071 (2006)
5. Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J.: Social coding in github: transparency and collaboration in an open software repository. In: CSCW 2012, pp. 1277–1286. ACM (2012)
6. Bird, C.: Sociotechnical coordination and collaboration in open source software. In: ICSM 2011, pp. 568–573. IEEE (2011)
7. Crowston, K., Wei, K., Li, Q., Eseryel, U.Y., Howison, J.: Coordination of free/libre and open source software development (2005)
8. Gutwin, C., Penner, R., Schneider, K.: Group awareness in distributed software development. In: CSCW 2004, pp. 72–81. ACM (2004)
9. Howison, J., Crowston, K.: Collaboration through open superposition: A theory of the open source way. MIS Quarterly **38**, 29–50 (2014)
10. Dourish, P., Bellotti, V.: Awareness and coordination in shared workspaces. In: CSCW 1992, pp. 107–114. ACM (1992)
11. Ko, A.J., DeLine, R., Venolia, G.: Information needs in collocated software development teams. In: ICSE 2007, pp. 344–353. IEEE CS (2007)
12. Damian, D., Izquierdo, L., Singer, J., Kwan, I.: Awareness in the wild: Why communication breakdowns occur. In: ICGSE 2007, pp. 81–90. IEEE (2007)
13. de Souza, C.R., Redmiles, D.F.: An empirical study of software developers' management of dependencies and changes. In: ICSE 2008, pp. 241–250. ACM (2008)
14. Blincoe, K., Valetto, G., Damian, D.: Facilitating coordination between software developers: A timely and efficient approach. Technical Report DCS-354-IR (2014)
15. Treude, C., Storey, M.: Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In: ICSE 2010, pp. 365–374. IEEE (2010)
16. Singer, L., Figueira Filho, F.M., Storey, M.A.D.: Software engineering at the speed of light: how developers stay current using twitter. In: ICSE 2014, pp. 211–221 (2014)
17. Corbin, J., Strauss, A.: Basics of qualitative research: Techniques and procedures for developing grounded theory. Sage (2008)
18. Gousios, G., Spinellis, D.: Ghtorrent: Github's data from a firehose. In: MSR 2013, pp. 12–21. IEEE (2012)
19. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D.: The promises and perils of mining github. In: MSR 2014, pp. 92–101. ACM (2014)
20. Parasuraman, A., Grewal, D., Krishnan, R.: Marketing research. Cengage Learning (2006)
21. Conway, M.E.: How do committees invent. Datamation **14**, 28–31 (1968)

The Diffusion of Pastebin Tools to Enhance Communication in FLOSS Mailing Lists

Megan Squire(✉) and Amber K. Smith

Elon University, Elon, NC, USA
{msquire,asmith90}@elon.edu

Abstract. This paper describes how software developers who use mailing lists to communicate reacted and adjusted to a new supplementary collaboration tool, called a pastebin service. Using publicly-available archives of 8800 mailing lists, we examine the adoption of the pastebin tool by software developers and compare it to the model presented in Diffusion of Innovation (DoI) theory. We then compare the rate at which software developers decided whether to accept or reject the new pastebin tools. We find that the overall rate of pastebin adoption follows the S-curve predicted by classic DoI theory. We then compare the individual pastebin services and their rates of adoption, as well as the reaction of different communities to the new tools and the various rationales for accepting or rejecting them.

Keywords: Open source · Pastebin · Email · Diffusion of innovations · Software development

1 Introduction

Software developers working in distributed teams, such as on free, libre, and open source software (FLOSS) projects, have historically used digital tools to communicate with each other about bugs, features, and decisions related to the project. Developers on these types of projects are also often geographically and temporally distributed, making the use of digital media a requirement for communication. Traditionally, the most common digital tool for software development communication has been the email mailing list. Many of these email mailing lists are publicly-viewable and archived for the long term, since FLOSS development relies on a certain level of openness in participation, transparency in decision making, and institutional memory. Apache Software Foundation projects, for example, are required to conduct all official project business on the mailing list (e.g. [1] [2]). Software developers can use email mailing lists to send each other long text artifacts for review, such as bug reports, code snippets, error logs, and the like.

However, when compared to newer social media web sites, email mailing lists can seem simplistic. As [3] explains, over time, more software development will be conducted by "social programmers" using web sites and apps designed for sharing artifacts, and mailing lists risk obsolescence. For example, while mailing lists can be expressed as a type of primitive social network [4], they are not nearly as expressive of social relationships as are microblogging services like Twitter or the pull request system of Github. For reputation management, email lacks the badging and voting

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 45–57, 2015.
DOI: 10.1007/978-3-319-17837-0_5

features of Q-and-A web sites like Stack Overflow. For collaborative editing, mailing lists are not as easy-to-use as a wiki or a shared code editor.

And yet, email persists as a pillar of "social programming". In the mid-2000s a class of web site was created to facilitate sharing source code via email. A site like this is called generically a "pastebin", after Pastebin.com, one of the first such sites. A pastebin is a web site that allows a user to paste in text and receive back a permanent short URL to the text that was pasted. Some pastebins even include syntax highlighting for common programming languages. This can be an appealing advantage when constructing an email with source code in it, or multiple attached log files, or long error logs, or complicated bug reports. Developers using a pastebin advocate that it enhances the utility of the email mailing list as a social communication tool: it makes sharing text easier and reading more efficient.

For this paper, our research questions center around diffusion of these pastebin tools on software development mailing lists, as follows:

- RQ1: What is the rate of diffusion for pastebins among FLOSS developers using mailing lists?
- RQ2: Does the rate of diffusion change between different variants of the innovation?
- RQ3: What are the stated reasons for and against adopting this innovation?

By studying the rates of diffusion of this pastebin tool, we can better understand how contemporary distributed software development is done, and whether certain tools designed to facilitate social programming on older communication tools will be adopted or not.

To answer these questions, Section 2 gives some background on the pastebin innovation itself and we review the tenets of classical Diffusion of Innovation (DoI) theory. In Section 3 we present our methods for measuring the diffusion of this innovation in the FLOSS developer community, specifically in its email mailing lists. In Section 4, we review the results of this analysis and present our findings for how and why the innovation diffused, including differences across tools. In Section 5 we discuss the implication of our method and analysis on our research questions. In Section 6 we explain limitations of our method and in Section 7 we make recommendations for future study.

2 Background

2.1 Pastebins

As the frequently asked questions (FAQ) document of one popular pastebin tool explains, the website "is mainly used by programmers to store pieces of sources [sic] code or configuration information, but anyone is more than welcome to paste any type of text. The idea behind the site is to make it more convenient for people to share large amounts of text online." [5] Examples of pastebins include: Pastebin.com, Github Gists, and Paste.org.

The innovation of a plain pastebin represents the unification of several previous ideas: a pastebin at this level is simply a very quick way to publish a text document on

the web and give it a shortened URL suitable for sharing. Pastebin.com requires no authentication (although that option is available), and the URLs can be set to expire in increments ranging from minutes to "never". The goal is to simplify the publication process from a complicated one involving file transfer and permissions-setting, to one involving only pasting into a web form.

More recently, there is also a class of web sites called an online IDE (integrated development environments) which not only allow the developer to paste in code and provides a link back, but also allows this code to be modified and run in the browser. Examples of online IDEs include CodePen, jsFiddle, and JS Bin (for testing JavaScript code), SQL Fiddle (for testing SQL), PhpFiddle (for testing PHP code), and CodePad (supports a variety of languages).

We should note that because of the anonymity and convenience provided by pastebin sites (and even online IDEs), they have also been used for non-software related purposes, including some illegal or of questionable legality (for example sharing of password lists, posting stolen credit card numbers, high volumes of spam link farming, and the like). See [6] for a description and timeline of some of the more notable illegal activities on Pastebin.

Another important aspect of all pastebin sites is the longevity of the provided URL. Each pasted document is initially given a unique URL, but if the document is allowed to expire, this URL will become a dead link ("link rot"). Pastebin sites have also gone defunct (perhaps due to their site administrators not expecting the high volume of "alternative" uses of their sites, see [7] for details about the expense involved in eradicating spam on Pastebin.com) and this means any links are also defunct.

2.2 Diffusion of Innovations

We are particularly interested in how these pastebin tools diffused into common usage among software developers. How innovations diffuse (diffusion of innovations, or DoI) has a rich literature, stemming from the work of Everett Rogers [8] who laid the foundation for how to systematically study the process that a new idea goes through as it becomes accepted or rejected by a community. Among Rogers' contributions was to document the "S-curve" that innovations typically go through on their way to becoming accepted or diffused. If we plot time on an X-axis and usage/acceptance of an innovation on the Y-axis, the typical path of an innovation over time will look something like the letter "S". See Figure 1 for an example. The steepness or shallowness of the S will be interesting to the diffusion researcher (as will the lack of an "S" shape, if the innovation was not successful in diffusing at all).

DoI research also tends to be concerned with the processes that led to that particular rate of diffusion, including characteristics of the community (or individuals) that would affect the diffusion, such as the compatibility of the innovation with the community's values or the effectiveness of a change agent to champion the innovation. Diffusion can be quite complex, especially with technologies that are themselves complex or networked systems [9]. Entities (governments, companies or individuals) with a stake in getting their particular innovation adopted will attempt to understand the myriad variables that can affect the rate or likelihood of diffusion, in order to make that diffusion process more effective.

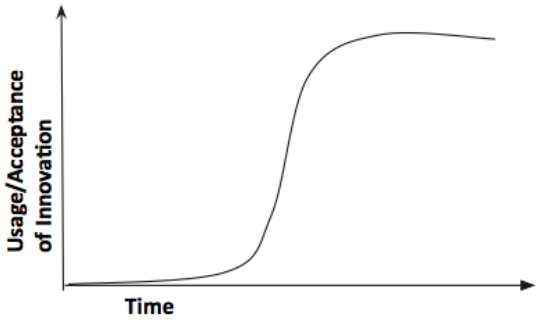

Fig. 1. Typical Diffusion of Innovations curve (modeled after Rogers, 1964)

Once diffusion is determined to have occurred, interesting questions may center around the following: describing the initial discovery of the tool by early adopters, the steps taken by early adopters to encourage others to use the tool, the rejection and refusal by some community members to use the tool, and the expectation-setting and rule-setting by the community governing use of the new tool.

For this project, we are primarily interested in doing the foundational work of calculating the rate of diffusion of pastebin tools within the community of social programmers who use mailing lists to communicate. We also take the first steps toward understanding how early adopters encouraged later adopters to use the pastebin innovation.

3 Methods

3.1 Data Collection

To answer our research questions, we needed first to retrieve the count of times each pastebin tool was mentioned on software development mailing lists over time. The result will be plotted as a rate-of-diffusion curve. To get this data we first identified a source for searching software development mailing lists by keyword. All of our mailing list data came from a publicly-available email aggregation web site called MarkMail.org. MarkMail provides a search interface for approximately 70 million emails from more than 8800 software development mailing lists, in the time period 1992-2014. MarkMail allows broad keyword searches, and also allows searching within typical email headers (e.g. from, subject, and list address).

With MarkMail as a reliable source of email data, we then had to figure out which words were being used to describe pastebin services. The web site Pastebin.com was begun in 2002, and the first mention of either the Pastebin.com web site or the generic noun ("pastebin") on a mailing list was on May 11, 2003 [10]. Though Pastebin.com was probably the first in widespread, public usage [7], there are many more pastebin-style websites in existence now.

The generic noun "pastebin" is still used to refer to both the site type ("Use a pastebin to post your code!") and the actual text having been pasted ("I can't find that pastebin you sent"), but other, newer pastebin tool names are often used generically as well (for example, "I'm going to create a gist" or "Did you get my dpaste?"). We therefore needed to search for full or partial web site URLs (e.g. "pastebin.com", "gist.github.com"), and

the corresponding generic nouns ("send me a pastebin" or "as you can see from my gist"). To answer our first research question about the rate of diffusion we needed to count all mentions of pastebins, whether by URL or by generic noun.

To identify relevant pastebin tools to use as search terms, we constructed a master list of 38 pastebin web sites and online IDEs that are used in software development. We then searched for each tool in MarkMail to see which were most used by software development teams. The results are shown in Table 1.

Table 1. Top 12 Most Used Pastebin Websites on MarkMail (2003 - 2014)

Rank	Pastebin Website	Message Count	Rank	Pastebin Website	Message Count
1	Pastebin.com	64009	7	Codepad	847
2	Github Gists	39557	8	Paste.org	467
3	Pastie	13397	9	Codepen	224
4	jsFiddle	10536	10	IDEone	196
5	Dpaste	6740	11	SQLFiddle	54
6	JS Bin	2416	12	all others	< 50 ea.

Most of the tools have the same common name as their URL domain (e.g. jsFiddle). However, two of the pastebin tools, Github Gists and Paste.org, were harder to search for as generic nouns since their relevant stem is already a word in common English usage. Table 2 shows the results for paste and gist as generic nouns, versus their URL-specific versions.

Table 2. Comparing the message count for 'gist' and 'paste' as words and as URLs

Original Keyword	Message Count	New Keyword	Message Count
gist	68706	gist.github	39557
paste	674760	paste.org	467

To fix these problems, we decided to search for Github Gists using the partial URL gist.github only, and we limited results for the "Paste.org" website to only those messages that included the partial URL paste.org. We recognize that, by doing this, we may have missed some instances of *gist* or *paste* used generically to refer to the pastebin tool. We also implemented some basic data cleaning procedures. One mailing list (com.googlegroups. jquery-br) used the words JS Bin and jsFiddle in nearly every signature line on over 50,000 messages sent. We removed these.

For each of the four most frequently used pastebin sites shown in Table 1 (more than 10,000 mentions), we collected the data for usage over time (message counts by month and year). We wrote scripts to download the counts for each term. The scripts are available on Github for anyone to use [11]. The next section describes our analysis of this data.

3.2 Data Analysis

We have created several charts showing the count of messages with each of the top four pastebin sites mentioned. Figure 2 (next page) shows the overall percentage of

messages mentioning any pastebin site from May 2003 until February 2014. The percentage of mail mentioning any pastebin tool ranges from less than 0.1%, but currently hovers around 0.5% of all mail.

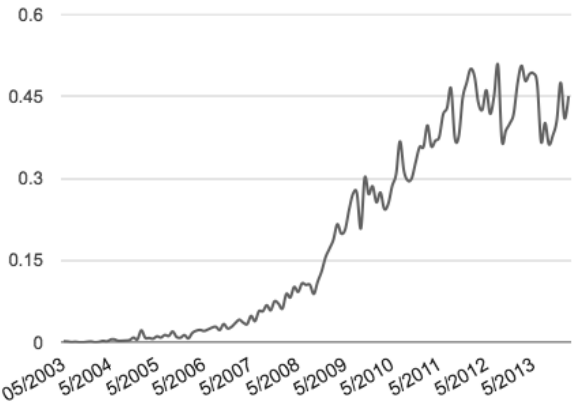

Fig. 2. Percent of mailing list messages making reference to any pastebin, 2003 - 2014

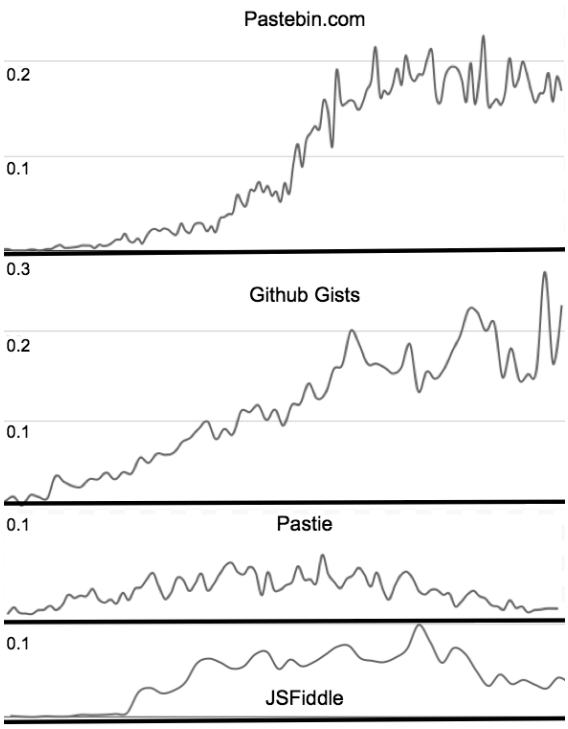

Fig. 3. Comparison of pastebin tool references as a percentage of all mail sent, 2003 - 2014

Figure 3 shows each particular diffusion line for the different pastebin tools. The X-axis (time) for each sub-graph in Figure 3 differs by when that pastebin site was created. For example, the first mention of Pastebin.com itself was in May 2003 [10], whereas the first mention of Github Gist was in July 2008 [12]. Each sub-graph in Figure 3 has been smoothed, and each Y-axis shows the keyword as a percent of all mailing list messages in MarkMail (to take into account natural variations in email usage over time by teams in general). These range from <0.1% to nearly 0.3%.

4 Discussion

Figure 3 shows some of the same steep increases and "S-curve" that is expected in DoI theory for the largest pastebin sites (pastebin.com, Github Gists), and to a smaller extent with jsFiddle. Pastie is comparatively flat throughout its existence. In this section we describe each of the diffusion curves individually, compare their shapes, and give reasons for the differences.

4.1 Pastebin.com Discussion

The curve for diffusion of Pastebin.com is shown in the top bar of Figure 3. The web site for Pastebin.com claims that it was started in 2002, and indeed the first Pastebin.com snapshot [13] on the Wayback Machine [14] was taken on November 23, 2002. But the first mention of Pastebin.com on any of the email mailing lists was six months later on May 11, 2003 [10]. By November of 2003, or one year after its creation, some mailing list users were treating the use of Pastebin.com as a social norm (an expectation for doing business on the mailing list), and some were actually shaming "newbies" for NOT using it [15]:

```
> You need to get a website and post a link to the site instead
of posting a giant email
> with code that's not even indented and probably wrapped in
some places.

Agree with CP here - please don't post large code get some web-
space, or use www.pastebin.com; please don't send attachments;
please don't send your messages priority, request return re-
ceipts; please turn of [sic] vacation auto-responders, please
use plain text, and above all else, please don't top post.
(where's that weekly newbie email?)
```

In Rogers' description of the diffusion of innovations, early adopters often become the champions or advocates of the innovation. However, one thing that is interesting about the message in [15] is that not only does this admonishment happen relatively quickly in the life of the pastebin tool, but this particular list (php-general) is a mailing list specifically designed for all types of users, whereas the original place that the tool was diffused was a developer-specific list. New users ("newbies") are apparently expected to know about the existence of this tool very quickly.

Indeed, the "weekly newbie email" to which the email author refers was indeed sent out just the day before the exchange above [16], but it did not mention

Pastebin.com nor did it even encourage people to point to code on the web by using a URL. In fact, this idea was not officially recommended to newbies on that mailing list until April of 2004, five months after this user was admonished. [17]

Another interesting pattern we see in the diffusion of Pastebin.com is how rapidly "pastebin" becomes a generic noun, and how quickly copycat web sites spring up. The first mention of a copycat web site is about 13 months after Pastebin.com was first created, and about 7 months after the first use of 'Pastebin.com' on mailing lists [18]. The first mention of pastebin as a generic noun is in a February 2004 message [19] describing a new tool called "Trash: a pastebin application written using the Twisted framwork" [sic].

Eventually (certainly by 2009) there is some pushback in at least some portions of the developer community to using a pastebin on mailing lists. The common reasons given are, first, that emails with pastebin links are annoying to read because you have to click a separate link. As [20] begs,

> ...please (please please) don't use pastebin. Just include the output inline in the mail message. It is much easer[sic] to get at then.

Second, developers noticed that the pastebin URLs eventually time out or disappear ("link rot"), making them unreliable for long-term issue tracking. For example the developer in [21] explains,

> ...Pastebin is a useful tool for sharing information between people, but please do not reference pastebin URLs in Jira Tickets. Pastebin entries are not guaranteed to survive for any period of time, and they can and will disappear at some point. This means of [sic] you open a ticket and you put a backtrace in the pastebin, if the pastebin is cleared so is your backtrace.

4.2 Github Gists Discussion

Github introduced its Gists service on July 21, 2008, and the first mention of 'gist.github' on a mailing list was two days later [12]. Whereas Pastebin.com took six months to diffuse onto mailing lists, the fact that it took Gists two days to diffuse confirms the fact that pastebins were well-understood by developers at this point. Thus the Gists adoption curve tracks upward more quickly than Pastebin (Figure 3).

However, we still find evidence of some pro-innovation diffusion behaviors inside the developer community using Gists: developers encouraging others to use Gists when it is first introduced [22], new users commenting on the fact that they are trying Gists for the first time [23] developers or teams setting rules or community procedures around Gists [24][25], and new technical reasons for using Gists (e.g. the mailing list is set to reject attachments [26], or the mailing list software itself rejects messages with code in them [27]). Like with Pastebin.com, there is even occasional shaming when others do not use Gists [28].

Some developers also make the case for Github Gists as an improvement to the mailing list experience in terms of increased "social programming" power [23] as follows:

> I've never used pastebins before. I heard about them the other day when I got roundly booed for pasting some code into an irc window. Can someone tell me how they work?
>
> This lets someone see that file with syntax highlighting in a browser, instead of in an IRC window, etc.(…) I think it's kinda cool to be able to collaborate that way with someone who might not know much about git or source control tools in general, better than posting revisions (…) or sending patches around on mailing lists.

4.3 jsFiddle Discussion

jsFiddle represents an interesting case of a language-specific (JavaScript) online IDE which also provides static, sharable URLs. As such, jsFiddle does not have as broad a potential developer user base as Pastebin.com or Gists, both of which are language-neutral. Its adoption curve is therefore smaller than the other two (Figure 3). Also, jsFiddle is designed to help a developer code and test in the browser, so it will not be effective for error logs or other non-code messages. Finally, most of the mailing lists indexed by MarkMail are for FLOSS projects that are written in high-level, multi-purpose, non-browser languages (for example Java, Python, Ruby, C++), so they have little use for a language-specific interface-driven tool like jsFiddle. (We also ran into some issues with the way MarkMail collected data that affected jsFiddle, and these are discussed in the Limitations section.) Another study did note the popularity of jsFiddle on Stack Overflow [29], finding that it was the sixth most common domain linked in postings.

The first mention of jsFiddle on the mailing lists in MarkMail was in January of 2010, but it took a year for jsFiddle mentions to accelerate to more than 100 mentions per month (0.03% of messages). As we mentioned in Section 3.1, we had to clean the data to remove one particular mailing list (jquery-br) because a list-wide email signature line was artificially inflating the numbers for both JS Bin and jsFiddle. However it is worth pointing out that the reason that the list members put that request in their signature lines ("Use JSBIN.COM / JSFIDDLE.NET for code") in the first place was as a way of directing the community towards a preferred behavior. This was the same pattern that we noticed in Pastebin.com, and Github Gists as well: the developers themselves would decide to adopt the innovation and then attempt to encourage others to use it by modeling positive behaviors, social norming and rule-setting, or shaming. Here is an exchange between a new user and a more experienced developer regarding jsFiddle [30]:

> Maybe you can get your issue into jsFiddle?
>
> Thanks for the suggestion to use jsfiddle. It helped to quickly allow me to test out my solution. You can see it in action here...

5 Results

Our first research question asks what the rate of diffusion is for pastebins among software developers using mailing lists to communicate. The graph in Figure 2 shows the number of times popular pastebin tools were mentioned in messages to the 8800 mailing lists on MarkMail. The graph shows the classic DoI adoption S-curve: a slow ramp up, followed by a steeper period of increasing usage, which eventually levels out.

Our second question asks whether the diffusion curve differs between different variants of the innovation. Figure 3 shows different shapes to each of the top four pastebin tools. The "first mover" tools either took a long time to ramp up (Pastebin.com) or never quite took off (Pastie). The later-to-market tools were adopted quite quickly and enthusiastically by either a general audience (as with Github Gists) or even in a comparatively narrow niche market (as with jsFiddle).

Our third question is about the stated reasons for and against using this tool to enhance communication on mailing lists. In reading the emails in which developers try to convince their colleagues to adopt the pastebin innovation, they rely on a number of persuasive techniques. First, some developers issue simple entreaties to make everyone's reading experience better. These arguments center on the anti-social aspects of asking for help while simultaneously pasting in thousands of lines of code into a message. We also found pro-pastebin arguments that address the technical downsides to mailing lists, namely that they are overrun with spam and therefore may have been configured to have length limits or rules disallowing attachments.

But we find the most interesting group of arguments are those which claim that a pastebin or online IDE will actually make the programming process more social and therefore more effective. This is especially true in the case of Github Gists (which can be forked, and pull requests issued) and in the case of jsFiddle (which can be edited and the new URL re-sent). In this case, a pastebin is more than a simple link-to-text scheme; it actually adds a layer of social enhancement that did not exist with simple mailing lists.

We also found some anti-pastebin sentiment. A number of developers point out the negative side effects of linking to code rather than archiving it inside the message. Some point out the inconvenience of having to click a link, and many more mention the risk that the link may be broken over time and is therefore unreliable as a record of what has been done in the code and why. However, based on the diffusion curves, this argument does not seem to have much "won" on developer mailing lists as a whole.

6 Limitations

Definition of Diffusion. Our method of calculating the diffusion rate may differ from the traditional interpretation of "diffusion" because of our use of message counts and not sender counts in our calculation of the adoption rate. In the traditional DoI literature, diffusion is usually defined as "number of people who adopt an invention". In the realm of email, adoption can be described as the number of senders who mention the tool, or it could be the number of times (emails) a sender mentions the tool. Table 3 outlines the differences in calculating diffusion using these two different methods.

Table 3. Two different methods of counting diffusion of pastebins on mailing lists

Method 1	Method 2
pm = count of messages mentioning a tool	*ps = count of senders mentioning a tool*
m = count of all messages	*s = count of all senders*
diffusion=pm/m	*diffusion=ps/s*

In this paper we choose to measure *pm/m* (method 1) rather than *ps/s* (method 2). We determined that it is actually more accurate to count messages rather than senders because for example, a single sender (e.g. "bugzilla@redhat.com") may actually represent an unknown number of other developers on the project. For example, suppose five different developers submit their bugs to a Bugzilla bug-tracking system, and they all used a pastebin for the error log or code sample that they are attaching to the bug tracking email. These five bug reports will all be forwarded to the mailing list as a single sender: bugzilla@company.com. Our method recognizes these messages as representing five instances of pastebin use, rather than just one. We also acknowledge that it is considerably easier to measure messages than to disambiguate senders (who may have multiple email addresses or different aliases).

Pre-Pastebin Code-via-URL Sharing. One other limitation of our study is that we do not have a good way for finding out how widespread the practice was of sharing code-via-URL prior to Pastebin.com. The [15] message shows that in November of 2003 there is at least some acknowledgement that it is possible (and may also be desirable) to create a web space for code snippet or log message rather than posting it in the email message. However, in this paper we do not examine how prevalent that practice was. It would be quite challenging to gather this count, since any URL would be a potential code location.

MarkMail Limitations. As a source for keyword searching through very high volumes of emails, MarkMail is a good (but not perfect) source. We recognize that MarkMail does not collect every FLOSS mailing list to begin with, and we acknowledge that some of its lists are not about FLOSS. More important for this project, we noticed that MarkMail may occasionally stop collecting from a mailing list abruptly, even though the list is still being used. We had some issues with MarkMail stopping the collection of one small mailing list in June of 2013. This abrupt stoppage affected our graph for jsFiddle since that one, small mailing list did include a very high number of jsFiddle users. (We turned in a support ticket to MarkMail about why they dropped the "jsknockout" mailing list from its collection, but it has so far gone unanswered.) Still, for searching a large number of lists over a long period of time, MarkMail is useful.

7 Future Work

We are excited about some avenues of future work, particularly in studying pastebin tools and the innovation curve on two additional sources of FLOSS social programmer communication: IRC and Stack Overflow. These two tools are heavily used by

active social programmers, especially in developing FLOSS and in distributed programming teams of all kinds. They are each very different ways of communicating, and have their own traditions, expectations, and culture. We suspect that the diffusion curves of the pastebin tool will look very different on each of these. Early evidence shows that pastebin tools caught on very quickly on IRC, in some cases producing an even sharper diffusion curve than on mailing lists. In contrast, we have encountered evidence of much more resistance to usage of pastebin tools in certain Stack Overflow postings but not in others. We suspect that this resistance is grounded in fear of link rot, as dead links are considered substantially more serious on a long-term reference site (which Stack Overflow aims to be) than on IRC and mailing lists. We also notice that some sub-communities within Stack Overflow are more accepting of pastebins (and online IDEs in particular) than other sub-communities. We are in the process of analyzing the IRC and Stack Overflow data and comparing it to mailing lists in order to provide evidence for how pastebins diffused on these other communication channels, and why.

8 Conclusion

Studying pastebin adoption on developer communication channels, mailing lists in this case, can further our understanding of how innovations diffuse in general, and in particular we can learn about how social media are impacting the traditional communication tools used by software developers. Will developers adopt a "social programming" innovation in order to enhance a tool that has already been in use for nearly forty years? In this paper we were able to confirm that the overall pastebin adoption does follow the S-curve of traditional DoI theory. Within particular tools, we find some differences in rates of diffusion depending on whether the tool being adopted was one of the earlier ones or one of the later ones. Most interestingly, we find that the reasons given for adopting the innovation include both very practical enhancements to the email communication medium, as well as attempts to improve the social aspects of collaborative coding.

References

1. Apache Software Foundation. Mailing Lists. http://www.apache.org/foundation/mailing-lists.html
2. Apache Software Foundation. Project Management Committee Guide. https://www.apache.org/dev/pmc.html#mailing-list-naming-policy
3. Storey, M.A., Singer, L., Cleary, B., Filho, F.F., Zagalsky, A.: The (R)Evolution of social media in software engineering. In: Proceedings of the on Future of Software Engineering, FOSE 2014, pp. 100–116. ACM (2014)
4. Bird, C., Gourley, A., Devanbu, P., Gertz, M., Swaminathan, A.: Mining email social networks. In: Proceedings of the 2006 International Workshop on Mining Software Repositories, pp. 137–143. ACM (2006)
5. Pastebin.com. Frequently Asked Questions. http://pastebin.com/faq#1
6. Brian, M.: Pastebin: How a popular code-sharing site became the ultimate hacker hangout. The Next Web, June 5, 2011. http://tnw.to/1CUpN

7. Kelion, L.: Pastebin to hire staff to tackle hackers' 'sensitive' posts. BBC News – Technology, April 1, 2012. http://www.bbc.com/news/technology-17544311
8. Rogers, E.M.: Diffusion of Innovations, 5th edn. Free Press (2003)
9. Lyytinen, K., Damsgard, J.: What's wrong with diffusion of innovation theory? the case of a complex and networked theory. In: Proceedings of the IFIP TC8 WG8.1 Fourth Working Conference on Diffusing Software Products and Process Innovations, pp. 173–190. Kluwer (2001)
10. Chapman, W.B.: PEAR DB_DataObject. On Pear-general list (2003). http://markmail.org/message/elidvojuxpb74kps
11. https://github.com/amberksmith/pastebin_data
12. Rasmussen, K.: Re: RFC: associations with :accessible => true should allow updating. On Rubyonrails-core mailing list, July 23, 2008. http://markmail.org/message/hhnacladvbfmn2ac
13. Internet Archive Wayback Machine Browse History for http://pastebin.com, http://web.archive.org/web/20020901000000*/, http://pastebin.com
14. Internet Archive Wayback Machine. http://web.archive.org
15. Khalid, B.: Re: BTML 2.0 released!!! On PHP-general list, November 6, 2003 http://markmail.org/message/7o7agkvsxwa4khsy
16. Unknown. [Newbie Guide] For the benefit of new members. On PHP-general mailing list, November 5, 2003. http://markmail.org/message/bvg6bqpthu6nhorf
17. Kumar, M.S.: [Newbie Guide] For the benefit of new members. On PHP-general mailing list, April 11, 2004. http://markmail.org/message/xmzgihotajd5ycar (last accessed April 29, 2014)
18. Wells, C.: Overload() problem. On PHP-general list, December 22, 2003. http://markmail.org/message/67vqonuzqf46qfje
19. Stepniewski, L.: Twisted Weekly News #11. On Twisted-Python mailing list, Feburary 17, 2004. http://markmail.org/message/tuqdi3ngd6jbngxi
20. Brown, N.: Re: 2 drives failed. On Linux-Raid-Vger list, January 29, 2010. http://markmail.org/message/lg6ohoycbiadu6j4
21. Rice, K.: Jira and the PasteBin…[Please READ ME]. On Freeswitch-users list. http://markmail.org/message/76gjydtvymg34sfc
22. Brown, G.: Re: Where to upload a ruby script to share it? On Ruby-lang Ruby-talk list, August 4, 2008. http://markmail.org/message/vaai47hafrjl3jgn
23. Stejerean, C.: Re: [Chicago] DVCS Workflows? On Python Chicago list, November 20, 2008. http://markmail.org/message/trj3b2r6e5726bdv
24. Kaiser, F.J.: [BP #4342] Announcement: BP-Testcase Page (html). On Blueprintcss list, January 25, 2011. http://markmail.org/message/d45yagbaj6d7vwmv
25. Leeming, C.: How to use the django-users mailing list properly and doing your homework. On Django-Users list, June 30, 2011. http://mark.mail.org/message/jl5lkq6sp6juw5vh
26. Ryaboy, D.: Re: java.lang.OutOfMemoryError when using TOP udf. On Apache Hadoop Pig User list, November 21, 2011. http://markmail.org/message/4jmb54whfvbenyqy
27. Rowe, S.: Re: Implement price range filter: DataImportHandler started. Not Initialized. No commands can be run. On Lucene Solr-User list, Feburary 14, 2013. http://markmail.org/message/3jguuhkgqdztynht
28. Demeshchuk, D.: Re: riak on one node. On Riak-Users list, October 1, 2012. http://markmail.org/message/mid5slijhzyp7xt3
29. Gomez, C., Cleary, B., Singer, L.: A study of innovation diffusion through link sharing on stack overflow. In: Proceedings of 10th Working Conference on Mining Software Repositories, pp. 81–84. ACM (2013)
30. Rpn. Re: Noob Question: Trouble binding computed value to table header. On Knockoutjs List, September 27, 2012. http://markmail.org/message/m43utkaz4eo5xffa

Examining Usability Work and Culture in OSS

Mikko Rajanen(✉) and Netta Iivari

Department of Information Processing Science, University of Oulu, Oulu, Finland
{mikko.rajanen,netta.iivari}@oulu.fi

Abstract. Organizational culture has been recognized as an influential factor affecting the successes and failures of usability work in organizations; however, there is a lack of research on organizational culture in open source software (OSS) development. This paper shows that there are different kinds of cultures in OSS development projects and builds propositions on the relationship between culture and usability work in OSS development projects. Partly those are derived from the literature, partly from an exploratory empirical inquiry. We speculate whether there is an ideal culture type for usability work in OSS development or whether usability work should be modified to fit the different cultures of OSS development projects.

Keywords: Open source software · Usability · Culture · Empirical study

1 Introduction

This paper examines usability work and organizational culture in the context of open source software (OSS) development. Usability work includes usability activities relating to analysis, design and evaluation that aim at making systems and products usable (e.g. [1,2,3,4]). The introduction of usability work into software development in general (e.g. [1,2]) and OSS development in particular [3,5,6,7,8,9] is challenging. In OSS projects usability has traditionally been neglected, as OSS developers have traditionally "scratched their own itch" and usability in the sense of ease of use has not been a major concern. Yet, nowadays many OSS solutions have attracted a large amount of users who do not want to participate in OSS development, but only to use the OSS. Thus, usability of OSS and usability work in OSS development have become crucial.

This paper argues that there are different kinds of cultures in OSS development projects as well as stipulates the role culture may play in the introduction of usability work into OSS development. The influence of organizational culture on usability work has been brought up in the literature (e.g. [1,2,11,12,13]). It has been argued that usability work should be compatible with the organizational culture in order to succeed [1,2,9,10,13]. As to OSS development, however, literature on the matter is very scarce. It has been brought up that OSS development projects have different kinds of cultures [6], OSS development culture may be in conflict with usability work [6,7,8] and usability activities should be tailored to fit the OSS development philosophy and culture [7,8,10]. Yet, no empirical research has been reported.

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 58–67, 2015.
DOI: 10.1007/978-3-319-17837-0_6

This paper initially inquires the cultural context of OSS development projects and speculates on the relationship between usability work and culture in OSS development. Culture (as defined in section 2) is approached here through the lens of the competing values model that is a widely used model for culture studies (e.g. [1,14]). Although the model has originally been developed for explaining differences in the organizational effectiveness literature [15], the value orientations in the model seem relevant also in OSS development. Moreover, the model has already been applied in related research on usability work in commercial software development [1]. The paper reports some exploratory research findings gained during a research program where attempts for introducing usability work into OSS development have been organized. Four OSS case projects are discussed in this paper.

This paper is structured as follows. The next section addresses the concept of culture and reviews related research addressing the relationship between usability work and culture. The third section presents the research method used in the empirical studies and the empirical results. The last section discusses the implications of the findings as well as their limitations and paths for future work.

2 Literature Review

Culture has been the topic of study within numerous disciplines, while in cultural anthropology it has been the main focus. The discipline studies humans as cultural beings, assuming that 'man is a symbolizing, conceptualizing, meaning-seeking animal' [16]. Although there are several hundred definitions of culture within the discipline, some are more prominent than others. A famous definition by Geertz is the following: "Man is an animal suspended in webs of significance he himself has spun, I take culture to be those webs, and the analysis of it to be therefore not an experimental science in search of law but an interpretive one in search of meaning" [16]. Later on, the study of culture has spread to different disciplines; including also Information Technology (IT) related disciplines. Some studies have even addressed the relationship between usability work and organizational culture.

In such studies, one can identify two strands: studies discussing the relationship between 'usability work and engineering culture' and studies discussing the relationship between 'usability work and a particular organizational culture'. The first strand argues that there are discrepancies between the cultures of engineers and usability professionals, and cultural change is needed for solving these discrepancies. Engineers need to cultivate their work practices [17], but also usability experts should try to think and work like engineers to minimize the problem of cultural differences [10,13]. The latter strand, on the other hand, discussing the relationship between usability work and particular organizational cultures, maintains that the introduction of usability work likely succeeds if it is customized to the existing culture as for usability work there is no 'one size fits all' [1,2,10]. One should understand the particular usability myths and values that define the usability culture of the organizations [11]. Obstacles for usability include prevalent myths, attitudes, beliefs and incentives [13]. Usability myths and values should be presented and openly discussed to succeed [11].

On the other hand, some studies do not recommend understanding the existing usability culture of the organization, but instead present 'usability culture' as an ideal

state where to aim at [12]. 'Strategic usability' is presented as an ideal state that means embedding usability in the organizational processes, culture and roadmaps [18]. 'Full scale usability' involves a major cultural transformation for an organization and a paradigm shift for practitioners. There is a need for a cultural change from technology and engineering centric views, but such change may cause resistance [19].

There are also studies that have shown that there are different cultures in development organizations that may have implications on usability work [1,20]. Certain cultural characteristics have been associated with certain usability work characteristics in organizations. Four 'usability cultures' have been identified that do not describe ideal situations, but current states of affairs in studied cases. The characteristics of usability work identified seemed to be compatible with the cultural characteristics [1]. Hence, the study recommends modifying usability work to fit the existing culture, proposing a cultural compatibility hypothesis (see [1,2]).

Based on the literature, it seems that there may be a cultural clash or conflict between usability work and engineering culture as well as between usability work and culture of some particular development organizations. To solve the problem of cultural clash or conflict, researchers suggest that: 1) Usability work should be modified to fit the engineering culture; 2) Usability work should be modified to fit some particular organizational culture; 3) Engineering culture should be modified so that usability is appreciated and 'usability culture' as an ideal state can be achieved; or 4) The organizational culture in question should be modified so that usability is appreciated and 'usability culture' as an ideal state can be achieved. There is variety in the culture conceptions in these studies as well as in the assumptions concerning the relationship between culture and usability work. Equipped with these tools, we examine studies addressing usability work and culture in OSS development.

Although during the past decade a huge amount of research on OSS development has been produced, there is a lack of research addressing the cultural context of OSS development. Nevertheless, it has been pointed out that there is variety in the OSS development community cultures, depending on the software under development, the size of the development community and the underlying business model [6]. However, empirical research on culture is lacking. On the other hand, researchers have already described the general characteristics of OSS development culture, bearing some resemblance with the description of engineering culture discussed earlier. OSS development culture has been characterized by passion and technical rationality: there are people passionate about the OSS they are developing and for them it might be difficult to empathize with users who do not have the similar level of technical knowledge and skills [6]. In OSS development culture, the interest is in scratching one's own itch and in finding technical solutions; there is no particular interest in understanding 'the user', but to show one's worth in practice [7]. The culture is described as developer-centric and merit-based that values technical skills and knowledge above all. In this kind of a cultural context it might be difficult for usability experts to gain merit [8].

Incorporating usability into this kind of a cultural context is a challenge [5]. However, the cultural clash is not only between usability and OSS development, but between corporate usability processes and OSS development [8]: decentralized and engineering-driven OSS development does not fit very well with corporate usability processes. It is argued that usability methods should better fit this culture [8]. Usability people should understand the cultural context they are entering into. Trust building

and showing merits are key concerns. Different strategies may be utilized. One may try to establish authority and trust by showing ones competence (in usability) with facts and data, or by trying to slowly integrate into the community [6].

This paper characterizes particular OSS development projects from the viewpoint of usability work and the cultural context and offers some initial propositions on the relationship between usability work and the cultural context. The cultural context will be addressed by using a competing values model as a sensitizing device. The model has been widely used in exploring organizational culture in IT research (e.g. [1,14,15,21]. It categorizes cultures based on value orientations in organizations. The model includes two axes that reflect the different value orientations: change vs. stability, and internal focus vs. external focus. Change emphasizes flexibility and spontaneity, while stability emphasizes control, continuity and order. Internal focus highlights integration and maintenance of the existing system, while external focus highlights competition and interaction with the organizational environment [21]. From these two dimensions one can identify four primary types of culture: group, adhocratic, hierarchical and rational. Usually organizations have features of all of them, while one usually dominates. Within the **group culture type** the emphasis is on flexibility and internal focus. The values are the sense of belonging, trust, participation, openness, teamwork, and the sense of family. Within the **adhocracy culture type** the emphasis is also on flexibility, but with focus on external environment. The values are innovation, adaptation, creativity, growth, resource acquisition, and dynamism. Within the **hierarchical culture type** the emphasis is on control and internal focus. Coordination, stability, measurement, documentation, order and smooth operation are valued. Finally, within the **rational culture type** the emphasis is on control and external orientation. Planning, goal setting, efficiency, productivity, competitiveness and market superiority are valued. The competing values model was utilized to make sense of the cultural contexts of the involved case OSS projects. The model provided us a concrete typology to be used in the venture related to which no existing research was found.

3 Research Design and Empirical Insights

This research is part of a larger research program, started in 2007, in which suitable methods and models for introducing usability work into OSS development have been developed and experimented with. This research program follows the design science approach, which is about building artefacts for specific purposes and about evaluating how well they perform for their intended purposes [22]. The artefacts have been methods for introducing usability work into OSS development. They have been iteratively improved through experimenting with them in real-life OSS development projects. During such experimentation, material also for this paper has been collected. The research program comprises 13 usability case projects in the OSS development context between years 2007 and 2014. Four cases were selected for this paper: they characterize clearly the differences between OSS project cultures among the 13 OSS case projects.

In this paper, OSS development cultures and usability work are studied in four different OSS development cases. In each case, a different student team introduced usability activities into one selected OSS development project under the close

supervision and guidance of the researchers, and collected data related to these usability activities and the OSS development project. These cases are reported in more detail from different theoretical viewpoints in [3,4,9]. All students had a background of multiple theoretical and practical usability courses. They acted as usability specialists in the OSS cases. Each student team consisted of three to ten students working between 200 and 300 hours each during four to six calendar months in planning and carrying out the usability activities, communicating with the OSS project, following up the impact of usability activities, collecting data, and writing project reports. The collected research material includes, e.g., community website content, version changelogs, emails, internet relay chat (IRC) logs, forum messages and reports of the usability activities. In this paper this research material is analyzed from the viewpoint of the OSS development culture and usability work. The competing values model offered a sensitizing device, focusing attention to the divergent value orientations in OSS development projects.

3.1 Case 1

In case 1, the usability intervention was done by a usability team of five students acting as external usability consultants (cf. [23]). The usability team kept its distance from developers and community as planned and therefore did not try to get to know this OSS project in detail before the usability activities. The usability team conducted heuristic evaluation, cognitive walkthrough and usability testing for the OSS and reported the findings in a report that was sent to the core developers and mentioned in a forum post at the main discussion forum of the community. The developers acknowledged receiving the report and said that they would respond when they had discussed it internally. After three years, no contact by the developers has been made, the identified usability problems have not been fixed and there has not been discussion about any usability related user interface changes in the forums.

In general, this OSS development community did not have a rigid hierarchical structure. The core developers were easily accessible by the usability team, which was indicated by the discussion forums and the IRC channel logs. Moreover, based on the discussions in the communication channels such as email lists, discussion forums and IRC channels of the project, the community and the developers seemed to be quite open to new ideas, new features, and improvements, but only as long as they were suggested by someone who was already recognized as merited by the community. Neither the community nor the core developers were really interested in an external group of usability contributors even though they were open to new ideas from within their own community. This was again indicated by the emails and IRC messages between the developers and the usability team. Altogether, the general mindset among the developers was not very encouraging for usability: the suggested usability improvements in the forums by users had been frequently shot down by the developers as being either irrelevant or a subjective matter of taste. Altogether, based on this evidence, we suggest that this OSS development project shares similar features especially with the group culture type in the competing values model, with emphasis on flexibility and internal focus [21].

3.2 Case 2

Given the failure of introducing usability activities into an OSS project by following a traditional external usability consultant approach in case 1, in case 2, the researchers tried a new approach by getting the student team into a position of an internal usability team. In case 2, the usability team consisted of three students who followed the OSS project's IRC channels and discussion forums for some time and tried to get to know the ways of the project (e.g. how to communicate appropriately in the project's IRC channel and discussion forums, what were the development practices of this OSS project, who would be the best developer to contact regarding usability issues etc.) before letting themselves and their intentions known. According to initial observations collected by the team, the OSS case 2 project had no prior knowledge or training about usability. The usability team contacted the core developers and established contact with the lead developer. The usability team tutored the lead developer about the concept of usability and offered their assistance to all things related to usability, trying to get a legitimate position within the OSS project. The lead developer got interested in the possible benefits of better usability and identified several possible areas for usability evaluation. The usability team performed heuristic evaluation and usability testing for the OSS and was in close contact with the lead developer regarding their findings and possible redesign solutions, and also participated in discussions in the project's IRC channel. After the evaluations, the usability team wrote a report of the usability issues, which included also their suggestions for changes to fix the usability problems. This time the work of the usability team had an impact. The core developers included the suggestions of the usability team as part of the changes to be made to the next version of the OSS and also fixed them in the next version. The core developers also contacted the usability team later, asking for a new usability evaluation for the next version.

The OSS community seemed to have a loose hierarchical structure, which was indicated by discussion forum and IRC channel messages. The core developers were easy to contact. They were interested in usability contributions and gave a warm welcome to the usability team even though they were not at first certain what usability was, which was indicated by the IRC and email messages between the developers and the usability contributors. In general, the community and the developers welcomed everybody willing to contribute towards the common goal of the community in any way, which was indicated by the welcoming attitude in the discussion forum and the IRC messages. Therefore, based on the evidence collected, we suggest that this OSS development project shares similar features especially with the adhocratic culture type in the competing values model, with emphasis on flexibility and external focus [21].

3.3 Case 3

In case 3, a usability team of ten students started by searching and following multiple communication channels of an OSS project for a couple of weeks, in order to get to know the proper ways of communicating in these channels, the use of community specific terminology, the development practices, and the already raised and discussed usability issues. The usability team tried to gain a legitimate position by contacting the core developers and offering their usability expertise for a particular area of the

software, which had already raised some discussion as regards the complicated user interface and the difficulties in use. Extensive usability testing and heuristic analysis were performed. The usability team wrote several reports about usability problems and their suggestions for changes to the user interface to fix these issues. These reports were put available on the usability team's blog and advertised in the project's IRC channels and discussion forums. The usability team informed the core developers and the community through IRC and discussion forums about the future usability activities, the usability reports and the redesign mock-ups. The reactions within the core developers and the community were varied; one core developer was very supportive for the usability activities while the other core developers and the community ignored the usability team and the usability discussion it tried to raise. The suggested changes to the user interface have not been made.

This OSS development community had a rigid, multilayered hierarchical structure and the leading core developer acted as the benevolent dictator, who communicated mainly with his trusted core developers, which was indicated by the community website and the lack of direct communication channels to the leading core developer. Hence, the leading core developer was inaccessible to the usability team. In this kind of OSS development community it may take a lot of time and effort to gain merit and access to the inner onion layers of the project. This likely applies to usability specialists, too. The evidence also indicates that this OSS development community was not very open to new ideas, especially to those proposed by outsiders: the discussions within this community showed that the core developers and the community in general had rejected many usability and user interface improvements and had firm ideas on those matters by themselves. Thus, based on the evidence collected, we suggest that this OSS development project shares similar features especially with the hierarchical culture type in the competing values model, with emphasis on control and internal focus [21].

3.4 Case 4

In case 4, similarly with cases 2 and 3, a usability team of five students followed the case OSS project's IRC channels and discussion forums for some time and tried to get to know the practices of the project before introducing themselves and their intentions. The usability team conducted heuristic evaluation and usability testing. The usability team wrote a preliminary and final usability reports about the usability issues and their suggestions for changes to fix them. The preliminary usability report, delivered to the project's mailing list, resulted in active discussion and lots of interest. The final usability report was delivered to the wiki of the OSS project, where the developers commented it actively. In addition, the usability team also submitted code patches and level design work, including new user interface menus and a new tutorial for the OSS. These were also received positively and they were accepted into the code repository. Moreover, the work of the usability team was referenced in several commit messages and one commit message asked explicitly for input from the usability team. Furthermore, one of the members of the usability team was invited to the development team and given commit rights as a result of his work in the usability team, his contributions to the code and discussions, and his recognized skills as a user of the OSS.

This OSS community had the traditional onion style hierarchical structure, but the culture of the community was open for new ideas and innovations. The development team of this OSS project was a tight group, who promoted into their team only those contributors who had contributed high level code, bug fixes or designs for a long period of time and whose ideas were in line with the design philosophy of the developers and the community. The development team made all decisions after lengthy discussions, trying to achieve a consensus. The core developers were easy to contact and the team was actually quite open to new members, as was indicated by the invitation of the usability team member into the development team. The core developers were also interested in the contributions of the usability team, which was indicated by the discussion forum and developer IRC channel messages. The development team encouraged the usability team to reduce all unnecessary tedious actions in the OSS to make the use better. In general, the community and the developers welcomed everybody willing to contribute towards the common goal of the community, which was again indicated by the discussion forum and the IRC channel messages of this project. Therefore, based on the evidence collected, we suggest that this OSS development project shares similar features especially with the adhocratic culture type in the competing values model, with emphasis on flexibility and external focus [21].

4 Concluding Discussion

There is a lack of research on culture in the OSS development context; thus, this paper contributes by offering initial insights on the matter. The literature review showed interesting distinctions in the literature. 'Usability culture' in an ideal sense as well as in a sense of current state of affairs was brought up. Some studies recommended changing the culture to fit usability work, while others emphasized that usability work should be modified to fit the culture in question. Initial results of our inquiry into culture and usability work in OSS development projects were presented. The competing values model was used as a sensitizing device and the four OSS case projects were classified to represent adhocratic, group or hierarchical types of culture [21]. Our usability intervention succeeded only in the OSS projects showing resemblance with the adhocratic type of culture, while in the unsuccessful cases the culture types identified were hierarchical and group culture type. However, our results on the relationship between usability work and different culture types in OSS development projects are clearly inconclusive as there are numerous issues that may be affecting the results. Quantitative research is required for testing these initial findings.

However, one can still speculate on the relationship between usability work and culture in OSS development projects. Our findings could imply that **the adhocratic culture type is the most suitable culture type for usability work in the OSS development context**. Thus, the description of the adhocratic culture type (e.g. [21]) could offer guidelines for identifying an ideal culture type for usability work in OSS development. Then one could either target only OSS projects representing this ideal state of affairs or try to change the culture of OSS projects representing other culture types. However, criticism against this kind of conception of culture has been expressed: cultures should not be viewed as something that can be intentionally changed

[1,2], probably even more so in OSS development projects than in commercial development organizations, as OSS communities usually operate on voluntary basis.

On the other hand, an alternative interpretation of the findings could be that: **when aiming at introducing usability work into OSS development; it needs to be modified to fit the culture type**. This proposition assumes that in the adhocratic culture type our approach was suitable, while for other culture types more fitting approaches need to be figured out. Along these lines, we next propose what this cultural fitting could entail for different culture types, relying on the work of Iivari [1], who has offered recommendations on how usability work could be modified to fit the different culture types. We adapt this work to suit the OSS development context and suggest that in an OSS development project: 1) With the group culture orientation, the emphasis as regards usability work should be on communal decision-making, informal information sharing, training and teamwork; 2) With the adhocracy culture orientation, the emphasis as regards usability work should be on innovation, experimentation, risk taking, supporting teamwork, brainstorming and iteration; 3) With the hierarchical culture orientation, the emphasis as regards usability work should be on careful planning and rules, procedures, control and documentation; and 4) With the rational culture orientation, the emphasis as regards usability work should be on measurement and cost benefit considerations that reveal the rationale for usability (cf. [3,4]).

In this paper, we utilized the competing values model for making sense of the cultural context of OSS development projects. Although the model has been widely used in culture studies, other models exist in the literature (e.g., Hofstede's model of culture and organizations) and other methods can be used (e.g., ethnography). As regards these four cases, it might also be considered as a limitation that the usability specialists in these cases were students. On the other hand, students from the IT field actually act as fully fledged developers in many OSS projects - students may have both development skills and time at their disposal. OSS projects usually do not prioritize formal education, but instead value the ability to contribute something useful to the project. In the described cases, the OSS developers did not see their status as students as being any kind of problem. In addition, the chosen OSS projects may affect the results of this experiment and more research is needed.

References

1. Iivari, N.: Representing the User in Software Development - A Cultural Analysis of Usability Work in the Product Development Context. Interacting with Computers **18**(4) (2006)
2. Iivari, N.: Culturally Compatible Usability Work - An Interpretive Case Study on the Relationship between Usability Work and Its Cultural Context in Software Product Development Organization. J. of Organizational and End User Computing **22**(3), 40–65 (2010)
3. Rajanen, M., Iivari, N., Anttila, K.: Introducing Usability Activities into Open Source Software Development Projects – Searching for a Suitable Approach. Journal of Information Technology Theory and Application **12**(4), 5–26 (2011)
4. Rajanen, M., Iivari, N., Keskitalo, E.: Introducing usability activities into open source software development projects: a participative approach. In: NordiCHI 2012 (2012)
5. Bach, P., DeLine, R., Carroll, J.: Designers wanted: participation and the user experience in open source software development. In: Proc. CHI 2009, pp. 985–994 (2009)

6. Bach, P., Twidale, M.: Social Participation in Open Source: What It Means for Designers. Interactions **17**(3) (2010)
7. Bødker, S., Nielsen, L., Orngreen, R.: Enabling user-centered design processes in open source communities. In: Proc. Human Computer Interaction International, pp. 10–18 (2007)
8. Terry, M., Kay, M., Lafreniere, B.: Perceptions and practices of usability in the free/open source software (foss) community. In: Proc. CHI, pp. 999–1008 (2010)
9. Rajanen, M., Iivari, N.: Power, empowerment and open source usability. In: CHI (2015)
10. Rajanen, M., Iivari, N.: Open source and human computer interaction philosophies in open source projects – incompatible or co-existent. In: Proc. Academic Mindtrek (2013)
11. Bloomer, S., Croft, R.: Pitching Usability to Your Organization, Interactions (1997)
12. Catarci, T., Matarazzo, G., Raiss, G.: Driving Usability into the Public Administration: The Italian Experience. Int'l J. of Human-Computer Studies **57** (2002)
13. Mayhew, D.: Strategic Development of Usability Engineering Function. Interactions **6**(5), 27–34 (1999)
14. Leidner, D., Kayworth, T.: Review: A Review of Culture in Information Systems Research: Toward a Theory of Information Technology Culture Conflict. MIS Quarterly **30**(2) (2006)
15. Quinn, R., Rohrbaugh, J.: A Spatial Model of Effectiveness Criteria. Management Science **29**(3), 363–377 (1983)
16. Geertz, C.: The Interpretation of Cultures: Selected Essays, New York (1973)
17. Anderson, W.L., Crocca, W.T.: Engineering Practice and Codevelopment of Product Prototypes. Comm. of the ACM **36**(4), 49–56 (1993)
18. Rosenbaum, S., Rohn, J.A., Humburg, J.: A toolkit for strategic usability: results from workshops, panels, and surveys. In: Proc. CHI 2000, pp. 337–344 (2000)
19. Hutchings, A.F., Knox, S.T.: Creating Products - Customer Demand. Comm. of the ACM **38**(5), 72–80 (1995)
20. Mirel, B.: Product, Process and Profit: The Politics of Usability in a Software Venture. ACM Journal of Computer Documentation **24**(4), 185–203 (2000)
21. Denison, D.R., Spreitzer, G.M.: Organizational Culture and Organizational Development: A Competitive Values Approach. Research in Org. Change and Devel. **5**, 1–21 (1991)
22. Hevner, A.R., March, S.T., Park, J.: Design Research in Information Systems Research. MIS Quarterly **28**(1), 75–105 (2004)
23. Schaffer, E.: Institutionalization of Usability: A Step-by-Step Guide. A-W, Boston (2004)

Examples and Case Studies

On the Availability and Effectiveness of Open Source Software for Digital Signing of PDF Documents

Jonas Gamalielsson[1(✉)], Fredrik Jakobsson[1], Björn Lundell[1], Jonas Feist[2], Tomas Gustavsson[3], and Fredric Landqvist[4]

[1] University of Skövde, Skövde, Sweden
{jonas.gamalielsson,bjorn.lundell}@his.se,
fredrik-jakobsson@live.se
[2] RedBridge AB, Kista, Sweden
jfeist@redbridge.se
[3] PrimeKey Solutions AB, Solna, Sweden
tomas@primekey.se
[4] Findwise AB, Göteborg, Sweden
fredric.landqvist@findwise.com

Abstract. Digital signatures are important in order to ensure the integrity and authenticity of information communicated over the Internet involving different stakeholders within and beyond the borders of different nations. The topic has gained increased interest in the European context and there is legislation and project initiatives aiming to facilitate use and standardisation of digital signatures. Open standards and open source implementations of open standards are important means for the interoperability and long-term maintenance of software systems implementing digital signatures. In this paper we report from a study aiming to establish the availability and effectiveness of software provided under an open source license for digital signing and validation of PDF documents. Specifically, we characterise the use of digital signatures in Swedish Governmental agencies, report on the interoperability of open source and proprietary licensed software for digital signatures in PDF documents, and establish the effectiveness of software provided under an open source license for validation of digital signatures in PDF documents.

1 Introduction

With increased communication over the Internet involving information exchanged between different stakeholders within and beyond the borders of different nations, there is an increasing need to ensure the integrity and authenticity of such information (Kaur and Kaur, 2012; Roy and Karforma, 2012). Digital signatures (also known as cryptographic signatures) are important means to ensure that information received is authentic and that it has not been altered in transit. Application areas are for example E-commerce, E-governance, and E-learning. In the European context the EU has issued a directive for the establishment of a community framework that forms the basis for legal recognition of electronic signatures[1] (EC, 1999). This directive has been

[1] Electronic signature in this context is a broader term that includes digital (cryptographic) signatures.

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 71–80, 2015.
DOI: 10.1007/978-3-319-17837-0_7

adopted in legislation in order to facilitate the use of digital signatures in a number of EU countries including Sweden (SFS, 2000). There is also a Commission decision on the publication of generally recognised standards for electronic signatures (EC, 2003), which impacts on the standardisation in this field in Europe. More recently, the eIDAs regulation on electronic identification and trust services for electronic transactions was adopted by co-legislators to increase the support for cross-border interactions between businesses, citizens and public authorities (EC, 2014).

Challenges related to digital signatures include *interoperability* between software systems for digital signing of documents and validation of digital signatures in documents. It is of vital importance that different (both open source and proprietary licensed) software systems can interoperate effectively utilising open standards, protocols, and algorithms (Bird, 1998; Ghosh, 2005; UK, 2012). In fact, interoperability supports systems heterogeneity, thereby increasing options for organisations (Ghosh, 2005). Another challenge concerns *long-term maintainable* systems with effective support for digital signatures, since it is of fundamental importance that old digitally signed documents can be validated in contemporary software. Further, any software system needs to be maintained beyond the life-cycle of any specific provider through use of open standards and open source licensed software (Lundell, 2012; Lundell et al. 2011). Especially, there is increased risk for lack of long-term availability of both software and digital assets (e.g. documents) "if the commercial vendor of adopted proprietary software leaves the market" (Lundell et al., 2011).

The ***overarching goal*** of the study is to establish the availability and effectiveness of software provided under an open source license for digital signing and validation of PDF documents. We make three principal contributions. ***First***, we establish a characterisation of the use of digital signatures in Swedish Governmental agencies. ***Second***, we report on interoperability of open source and proprietary licensed software for digital signatures in PDF documents. ***Third***, we establish the effectiveness of software provided under an open source license for validation of digital signatures in PDF documents.

We focus on documents in PDF format since it is one of the most commonly used document formats and is widely deployed in both open source and proprietary licensed software. There is limited knowledge on the state of practice concerning use of digital signatures in Swedish public sector organisations. Further, knowledge is limited concerning details regarding availability, interoperability and effectiveness of open source licensed tools for digital signing and validation of PDF documents. To the best of our knowledge, this study is the first to report on such details.

For the first contribution, an investigation in Swedish Governmental agencies is of particular relevance given that Sweden is amongst the most IT-intensive countries in the EU (WEF, 2014). For the second and third contribution, the open source licensed tools iText (Itextpdf.com, 2015) and PDFBox (Apache.org, 2015) were used. Both software tools were used in combination with the Bouncy Castle Crypto API (Bouncycastle.org, 2015), which is also provided under an open source license. Those tools were selected since they have been identified as mature and are amongst the most widely adopted (in organisations in practice) and deployed open source libraries for creation, signing and validation of PDF files. Adobe Acrobat Professional XI Pro (Adobe.com, 2015) was selected and used since it is one of the most adopted and

deployed proprietary tools for PDF processing, and is provided by the company that initially developed the PDF format. Further, for the second contribution LibreOffice Writer and Microsoft Word were chosen for generation of test documents with the motivation that they are both representative examples of software tools that are widely adopted, deployed, and provided under an open source and proprietary license, respectively.

The rest of this paper is organised as follows. We present a background on digital signatures and software support for digital signatures (section 2). Thereafter we present research approach (section 3), results (section 4), analysis (section 5), followed by discussion and conclusion (section 6).

2 On Digital Signatures and Software Support

A digital signature is an implementation of an asymmetric cryptography to ensure the integrity and authenticity of a document. A digital signature has the same purpose as the traditional physical signature, which is to prove the origin of the document, so that the recipient does not have any doubt that it was actually created by the person who sent it, and has not been tampered (with meaning or accidentally) along the way. The scheme for creation of a digital signature generally consists of three algorithms: 1) a key generation algorithm, which selects and outputs a private key and a corresponding public key; 2) a signature algorithm, which produces a signature given a private key and a message; and 3) verification algorithm, which accepts or rejects the authenticity claim of a message given a message, public key and a signature (Kaur and Kaur, 2012; Lowagie, 2013; Roy and Karforma, 2012). Central parameter choices for the signature algorithm include: encryption algorithm, e.g. RSA and DSA; standard for cryptographically protected messages, e.g. CMS (Cryptographic Message Syntax) and CAdES (CMS Advanced Electronic Signatures); and cryptographic hash (or "message digest") function, e.g. the SHA (Secure Hash Algorithm) function family, MD5, and the RIPEMD (RACE Integrity Primitives Evaluation Message Digest) function family.

There are European project initiatives aiming to facilitate use and standardisation of digital signatures in the European context. One such example is the SD-DSS project (Joinup.eu, 2011) in which the European Commission has commissioned development of open source software for use by Member States and associated service providers to be able to complete the procedures and formalities that are necessary for conduction of activities with Member States' administrations within and across borders. Another example is the E-signatures standards initiative whose mission is to create a rationalised framework for electronic signature standardisation in the European context (E-signatures-standards.eu, 2013). Specific standards addressed include CAdES, XAdES (XML Advanced Electronic Signatures), and PAdES (PDF Advanced Electronic Signature Profiles). This effort will support the realisation of one of the items of the EC Action Plan related to eSignatures, and is a collaboration between CEN (European Committee for Standardisation), EC (European Commission), ETSI (European Telecommunications Standards Institute), and the AFNOR Group (the French representative within CEN and ISO).

PDF is a document format that initially was maintained by Adobe. The PDF specification is available free of charge since 1993. PDF version 7 was released in November 2006 (Adobe.com, 2006), and in July 2008 this PDF version became available as an ISO standard (ISO, 2008). In the ISO standard for the PDF format there are details concerning support for digital signatures (ISO, 2008; Lowagie, 2013). Since the publication of the PDF standard (ISO, 2008), specific versions for specific purposes have been developed and standardised by ISO, for example the PDF/A standard for archiving purposes (ISO, 2005).

PDF is implemented in a number of software applications. **iText** is a library for PDF generation written mainly in Java (Itextpdf.com, 2015), and was initially provided under the MPLv1 and LGPLv2 licenses. However, the license was changed to the AGPLv3 license on 5 Dec. 2009 with the release of version 5.0.0. The latest stable version is 5.5.3, which was released on 17 Sep. 2014. The library is widely adopted in applications that include functionality for creation of PDF documents and digital signatures (using the Bouncy Castle Crypto API). **PDFBox** is a library that provides an API in Java to handle PDF documents (Apache.org, 2015), including signing and validation of digital signatures (using the Bouncy Castle Crypto API). The latest stable version is 1.8.7, which was released on 19 Sep. 2014, and is provided under the Apache License v2. It is used by eID-DSS, which is Belgium's solution for digital signatures for the eID solution for handling PDF documents. **Bouncy Castle** is a lightweight cryptography API for Java and C# (Bouncycastle.org, 2015). The latest stable version is 1.53, which was released on 28 Sep. 2014, and it is provided under the MIT license. It is a popular library that is utilised by various other open source projects and tools (apart from iText and PDFBox), including the SignServer application framework (Signserver.org, 2015). **Adobe Acrobat XI Pro** (Adobe.com, 2015) is software that can be used to view, create, manipulate, print and manage PDF documents. The latest stable version is 11.0.09 (released on 16 Sep. 2014), and it is provided under a proprietary license.

3 Research Approach

As the first part of our approach we establish a characterisation of the use of digital signatures in Swedish Governmental agencies. The data collection is made easier to answer in Sweden, which has a very strict policy on governmental responses to requests for public documents. In Sep. 2014 we sent an email in plain text to 71 Governmental agencies (the 16 IT intensive Governmental agencies in the Swedish e-Delegation, a selection of 35 other Swedish Governmental agencies, and all 20 Swedish Provincial offices). The email contained six requests: 1) Examples of one (or several) digitally signed PDF documents created within the organisation and sent to another public organisation; 2) Examples of one (or several) digitally signed PDF documents created within the organisation and sent to a corporation (or other private organisation) or to a private individual; 3) Examples of one (or several) digitally signed PDF documents submitted to the organisation from another public organisation; 4) Examples of one (or several) digitally signed PDF documents

submitted to the organisation from a company (or other private organisation) or from a private individual; 5) The documents (i.e. documentation from identification of needs, evaluations, decisions, contracts, procurement documents and other related documents, etc.) that relate to that (or those) contract(s) and (or) development projects related to the software and the systems used for creating and managing digitally signed PDF documents in the organisation; and 6) The documents (i.e. agreements, regulations, policy, strategy, instructions and other documents) that regulate and describe how digital signatures (and software for digital signatures) are to be used in the organisation.

Second, we analyse the interoperability of open source and proprietary licensed software for digital signatures in PDF documents. This was done by generating and signing test documents using different software and signature settings that were subsequently validated using different software. Specifically, the test documents were created by enumerating all 42 combinations of a specific software for generation (LibreOffice Writer 4.2.6.3 *or* Microsoft Word 13), a specific software for signing (Adobe Acrobat XI Pro *or* iText 5.5.3 *or* PDFBox 1.8.7), a specific signature format setting for signing (CMS *or* CAdES for Adobe Acrobat XI Pro, CMS *or* CAdES for iText 5.5.3, CMS for PDFBox 1.8.7), and a specific hash algorithm setting for signing (SHA256 for Adobe Acrobat XI Pro, SHA1 *or* SHA256 *or* SHA384 *or* SHA512 *or* RIPEMD160 for iText 5.5.3, SHA1 *or* SHA224 *or* SHA256 *or* SHA384 *or* SHA512 *or* MD5 *or* RIPEMD128 *or* RIPEMD160 *or* RIPEMD256 for PDFBox 1.8.7). Software tools used for validation of each of the 42 test documents were Adobe Acrobat XI Pro, iText 5.5.3, and PDFBox 1.8.7.

Third, we analyse the effectiveness of software provided under an open source license for validation of digital signatures in PDF documents. This was done by validating different PDF documents using different software. Specifically, the PDF documents provided by the Swedish Governmental agencies and Provincial offices according to requests 1-4 were used. In addition, all digitally signed PDF documents from a large corpus of one million US governmental documents of mixed file formats randomly selected from the ".gov" domain (http://digitalcorpora.org/corpora/govdocs) were used (approximately 20% of those files were PDF documents, of which 156 were digitally signed PDF documents). Software tools used for validation of these documents were Adobe Acrobat XI Pro (primarily for validation of the documents provided by the Swedish Governmental agencies), three versions of iText (v1.1.0: the first version with support for digital signatures, v2.1.7: the last LGPL licensed version, and v5.5.3: the latest version), and two versions of PDFBox (v1.6.0: the first version with support for digital signatures, and v1.8.7: the latest version). iText v1.1.0 and v2.1.7 was used in combination with Bouncy Castle Crypto API v1.38, whereas iText v5.5.3 was used in combination with Bouncy Castle Crypto API v1.53 (the latest version). PDFBox 1.6.0 was used in combination with Bouncy Castle Crypto API v1.38, whereas PDFBox 1.8.7 was used in combination with Bouncy Castle Crypto API v1.53.

For the second and third part, custom made shell scripts were used to integrate and utilise the iText and PDFBox libraries for signing and validation of the PDF documents, and for extracting meta data (e.g. date for signing) from the documents.

4 Results

4.1 On Use of Digital Signatures in Swedish Governmental Agencies

After sending the email that contained the 6 requests for information to each of the 71 Swedish Governmental agencies and Provincial offices including reminders we received totally 39 responses with answers. However, only a few of these contained any of the requested documents. In total, 15 examples of PDF documents according to requests 1 through 4 (see section 3) were provided by 10 of the Governmental agencies (of which four were provided by Provincial offices). In total, eight documents were provided by the Governmental agencies (excluding Provincial offices) of which two were according to request 1, two according to request 2, one according to request 3, and three according to request 4. All seven documents provided by the Provincial offices were according to request 3, but all of these documents were physically signed documents that had been scanned rather than documents with digital (cryptographic) signatures.

Concerning requests 5 and 6 to Governmental agencies (excluding Provincial offices), one agency has provided documents, 33 agencies have stated that there are no such documents. Concerning requests 5 and 6 to Provincial offices, two have stated that there are no such documents, one has stated that there are no documents for request 5 but that documents for request 6 will be provided (at time of writing not yet received).

4.2 Interoperability of Software for Digital Signatures

All 42 generated and digitally signed PDF files could be validated successfully by iText 5.5.3 and PDFBox 1.87. Adobe Acrobat XI Pro failed to validate four of the 42 files that were signed using PDFBox 1.8.7 and the hash algorithms RIPEMD128 and RIPEMDF256. The reason for this is that those hash algorithms are not supported in Adobe Acrobat XI Pro.

4.3 Open Source Software Support for Validation of Digital Signatures

For the eight PDF documents provided by Swedish Governmental agencies, two documents were successfully validated by all the tested tools (in all versions). Three additional documents were successfully validated by all the tested tools (except iText v1.1.0, since the hash algorithm used for signing the document is not supported in this version). The remaining three documents could not be validated by any of the tested tools (in any version). One of these could not be validated since no digital signature could be found according to all tested tools, and the other two documents had a proprietary signature format that was not supported in any of the tested tools. Four of the five documents that could be validated were signed in 2014 and the fifth document was signed in 2013.

For the 156 signed PDF documents from the ".gov" domain, six documents could not be successfully validated using any of the tested tools (in any version). By attempting to validate the signatures in these six documents in Adobe Acrobat XI Pro it

was found that the signatures contained either “incorrect”, “unrecognized”, “corrupt”, or “suspicious” data. The remaining 150 documents could be successfully validated by all the tested tools (except iText v1.1.0, for which 20 additional documents could not be validated since the hash algorithms used for signing the documents are not supported in this version). The 150 documents that could be validated were signed in the interval 2000-2009 and the majority of documents were signed during 2008-2009, see Table 1.

Table 1. Number of signed documents per year for the 150 validated “.gov” PDF-documents

Year	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009
# doc	5	13	4	7	6	8	9	19	50	29

5 Analysis

First, from the results (section 4.1) it is clear that the use of digital signatures in Swedish Governmental agencies (including Provincial offices) is very limited. Very few Governmental agencies have provided any documents on how digital signatures for documents are created, managed, and used within the organisation. Few digitally signed documents have been provided (only eight in total, of which five actually contained valid signatures). Hence, in general there seems to be a lack of policy for and deployment of digital signatures and their use in Swedish Governmental agencies. Except from the Swedish law concerning qualified electronic signatures and requirements on qualified certificates and the issuing of such certificates (SFS, 2000), there are no nationwide regulations or recommendations concerning when and in what contexts digital signatures shall be used.

Second, from the results (section 4.2) it is evident that the open source licensed tools are at least as effective as the proprietary licensed tool Adobe Acrobat XI Pro for signing PDF documents and validating signatures in PDF documents, and that the tested tools to a very large extent are interoperable. The lack of support for two specific hash algorithms (RIPEMD128 and RIPEMD256) in Adobe Acrobat XI Pro in the interoperability test may not be surprising, since support for these is not required according to the ISO specification for PDF (ISO, 2008, p. 476).

Third, from the results (section 4.3) it is clear that an overwhelming majority of the 156 digitally signed PDF documents from the “.gov” domain could be successfully validated in all the tested tools. It can also be noted that documents as old as 15 years could be validated. This shows, from a long-term maintenance perspective, that it is possible to validate older digitally signed documents using contemporary software. It also shows that software older than the digitally signed documents can be used for validation, since the digital signatures in the five files provided by Swedish Governmental agencies are from 2013-14 and could be validated with iText v2.1.7 (released 7 Jul. 2009) in combination with Bouncy Castle Crypto API v1.38 (released 7 Nov. 2007) and with PDFBox v1.6.0 (released 1 Jul. 2011) in combination with Bouncy Castle Crypto API v1.38 (released 7 Nov. 2007).

We acknowledge that software solutions for digital signing of PDF documents may be combined with hardware modules for enhanced security. However, as the overarching goal of our study is to establish the availability and effectiveness of open source software for digital signing of PDF documents, the analysis of such solutions is beyond the scope of our study. For future research it is relevant to also consider other file formats and associated open source software, in other usage contexts.

One openness aspect to consider is the licensing conditions for the standards for cryptographically protected message formats and hash functions used by the tested software tools, since it impacts on the possibility to (legally) provide implementations of the standards under an open source license. A patent search shows that there are no disclosed patents in the IETF patents database[2] for the most recent version of CMS (IETF RFC 5652, issued in 2009). However, there are two disclosed patents for an earlier version of CMS (IETF RFC 2630, issued in 1999) concerning technology used by S/MIME, a standard for public key encryption and signing of MIME (Multipurpose Internet Mail Extensions) data. For CAdES (RFC 5126) there have been no patent disclosures. For the ETSI version of CAdES (TS 101 733 V.1.7.4) there are no disclosed patents in the ETSI patents database[3]. Hence, the latest versions of CMS and CAdES do not seem to be encumbered by patents, and are therefore possible to implement in open source software. However, there may still be undisclosed patents that have not been reported to IETF or ETSI. A patent search in the ISO patent database[4] on a standard (ISO/IEC 10118-3:2004) involving the SHA and RIPEMD hash function families shows that there are several disclosed patents. Hence, these hash functions may cause problems when implementing such standards in open source software from a legal standpoint. There may also be other undisclosed patents that have not been reported to ISO for this standard.

Concerning certain standards for digital signatures (e.g. CAdES) there are a number of profiles offering different protection levels for different user groups, including CAdES-BES (basic form), CAdES-X (extended), and CAdES-LT (long term). This proliferation into different variants for the same standard may imply increased complexity and effort when developing and testing software that implements the standard.

6 Discussion and Conclusion

Use of readily available and effective open source tools for digital signing and validation of PDF documents is a strategy that simplifies the implementation of digital signatures for an organisation and that promotes a long-term sustainable software system with associated communities. In particular, open source licensed solutions offer more flexibility concerning which software version to use and when to update to a new version, something which often is critical for the stability of systems implementing digital signatures. Further, use of open standards for digital signatures is of vital importance since they promote interoperability between (both open source and

[2] https://datatracker.ietf.org/ipr/
[3] http://ipr.etsi.org/
[4] http://www.iso.org/iso/standards_development/patents

proprietary licensed) software systems, and also ensure long-term sustainability of the digital signatures.

Concerning the current limited use of digital signatures in the Swedish context, we envisage that increased efforts on provision of infrastructure for digital signatures at national level will promote increased use of digital signatures. Further, complex software solutions for digitally signing documents may inhibit adoption of digital signatures and improved system support simplifying this process can promote broader adoption of digital signatures.

In conclusion, our study shows that there are open source licensed tools available for digital signing and validation of PDF documents that are at least as effective as the proprietary licensed tool Adobe Acrobat XI Pro. Further, it is shown that the tested (open source and proprietary licensed) software tools to a large extent are interoperable. It is also shown that there is very limited use of digital signatures for documents in the context of Swedish Governmental agencies. The findings from our study therefore make an important contribution to practice and policy.

References

Adobe.com: PDF Reference – Adobe Portable Document Format, Version 1.7 (2006). www.adobe.com/content/dam/Adobe/en/devnet/pdf/pdfs/pdf_reference_1-7.pdf (accessed January 5, 2015)

Adobe.com: Acrobat XI Pro (2015). http://www.adobe.com/products/acrobatpro.html (accessed January 5, 2015)

Apache.org: Apache PDFBox – A Java PDF Library (2015). https://pdfbox.apache.org/ (accessed January 5, 2015)

Bird, G.B.: The Business Benefit of Standards. StandardView **6**(2), 76–80 (1998)

Bouncycastle.org: The Legion of the Bouncy Castle (2015). https://www.bouncycastle.org/ (accessed January 5, 2015)

EC: DIRECTIVE 1999/93/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on a Community framework for electronic signatures. Official Journal of the European Union, L13/12, December 13, 1999

EC: Commission decision on the publication of reference numbers of generally recognised standards for electronic signature products in accordance with Directive 1999/93/EC of the European Parliament and of the Council. Official Journal of the European Union, L175/45, July 14, 2003

EC: REGULATION (EU) NO 910/2014 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC. Official Journal of the European Union, L257/73, July 23, 2014

E-signatures-standards.eu: e-signatures standards – making e-signatures easy (2013). http://www.e-signatures-standards.eu/ (accessed January 5, 2015)

Ghosh, R.A.: Open Standards and Interoperability Report: An Economic Basis for Open Standards. FLOSSPOLS, Deliverable D4, Maastricht, December 12, 2005. www.flosspols.org

ISO: Document management – Electronic document file format for long-term preservation – Part 1: Use of PDF 1.4 (PDF/A-1). ISO/TC 171/SC 2, ISO 19005-1:2005 (2005)

ISO: Document management – Portable document format – Part 1: PDF 1.7. ISO/TC 171/SC 2, ISO 32000-1:2008 (2008)

Itextpdf.com: iText – Programmable PDF Software (2015). http://itextpdf.com/ (accessed January 5, 2015)

Joinup.eu: Digital Signature Service (2011). https://joinup.ec.europa.eu/asset/sd-dss/description (accessed January 5, 2015)

Kaur, R., Kaur, A.: Digital signature. In: Proceedings of the International Conference on Computing Sciences, ICCS 2012, September 14-15, pp. 205–301 (2012). doi: 10.1109/ICCS.2012.25

Lowagie, B.: Digital Signatures for PDF documents (2013). http://itextpdf.com/book/digitalsignatures20130304.pdf (accessed January 5, 2015)

Lundell, B.: Why do we need open standards? In: Orviska, M., Jakobs, K. (eds.) Proceedings 17th EURAS Annual Standardisation Conference 'Standards and Innovation'. The EURAS Board Series, Aachen, pp. 227–240 (2012) ISBN: 978-3-86130-337-4

Lundell, B., Lings, B., Syberfeldt, A.: Practitioner perceptions of Open Source software in the embedded systems area. Journal of Systems and Software **84**(9), 1540–1549 (2011)

Roy, A., Karforma, S.: A survey on digital signatures and its applications. Journal of Comper and Information Technology (IJCIT) **3**, 45–69 (2012)

SFS: Lag om kvalificerade elektroniska signaturer. Statens författningssamling, SFS 2000:832, November 2, 2000

Signserver.org: SignServer – PKI by PrimeKey (2015). http://www.signserver.org/ (accessed January 5, 2015)

UK: Open Standards Principles: For software interoperability, data and document formats in government IT specifications. Cabinet Office, UK (November 1, 2012)

WEF: The Global Information Technology Report 2014, World Economic Forum, Geneva (2014) ISBN-13: 978-92-95044-63-0

A Systematic Approach for Evaluating BPM Systems: Case Studies on Open Source and Proprietary Tools

Andrea Delgado(✉), Daniel Calegari, Pablo Milanese, Renatta Falcon, and Esteban García

Instituto de Computación, Facultad de Ingeniería, Universidad de La República, Montevideo, Uruguay
{adelgado,dcalegar,pmilanese}@fing.edu.uy, {renafal,estebangrd}@gmail.com

Abstract. Business Process Management Systems (BPMS) provide support for modeling, developing, deploying, executing and evaluating business processes in an organization. Selecting a BPMS is not a trivial task, not only due to the many existing alternatives, both in the open source and proprietary realms, but also because it requires a thorough evaluation of its capabilities, contextualizing them in the organizational environment in which they will be used. In this paper we present a methodology to guide the systematic evaluation of BPMS that takes into account the specific needs of each organization. It provides a list of key characteristics of BPMS which are ranked by the organization and evaluated using test cases and quantitative criteria. We also present case studies of open source and proprietary BPMS evaluations following our proposal.

Keywords: Business Process Management Systems (BPMS) · Open source and proprietary BPMS · Evaluation methodology · Systematic approach

1 Introduction

Every organization executes daily operations to achieve its goals, applying certain mechanisms to enable continuous improvement. The business process (BPs) vision is the identification of the set of activities that are performed in coordination within an organizational and technical environment to achieve defined business goals [1]. It provides support for the definition, control and continuous improvement of business operation. In this context, Business Process Management (BPM) [1,2] offers a framework to support the business process lifecycle [1] from modeling, through developing, deploying, executing and to the evaluation of their execution. BPM Systems (Business Process Management System, BPMS) [1,3] arise as the technology response to support the BPs lifecycle. These platforms integrate several components that allow modeling processes, executing them, controlling business rules, defining execution measures and monitoring processes, among others.

There are several process modeling and execution languages with different backgrounds and abilities, such as Business Process Model and Notation (BPMN 2.0) [4], XML Process Definition (XPDL) [5] and Web Services Business Process Execution Language (WS-BPEL) [6], among others. Likewise, there is also a wide variety of

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 81–90, 2015.
DOI: 10.1007/978-3-319-17837-0_8

BPMS, both open source and proprietary, with different support levels for the defined solution. In addition, several open source products offers several business models such as community edition with limited functionality, fees for maintenance and support, enterprise versions, among others. To be able to compare features within different BPMS, it is necessary to provide an objective evaluation regarding the fulfillment of key technical features that should be provided, as defined in academia [7,8] and industry [9,10] studies. However, the selection of the most adequate BPMS for an organization depends not only on the technological support it provides, but also on the characteristics of the organization itself. The evaluation should also be guided by a systematic procedure to ensure the quality of the results and its repeatability.

In this paper we present a methodology for evaluating BPMS considering the specific needs of each organization. Our approach includes the definition of key activities to guide the evaluation and a list of key features that are relevant to this kind of systems. This methodology has been developed within our research group and has been applied for evaluating open source and proprietary BPMS in several projects[1]. To illustrate the approach, we present results from these projects which constitute both a validation and assessment of our proposal and a contribution to knowledge regarding the capacities of selected BPMS technologies.

The rest of the article is organized as follows. In Section 2 we discuss related work and in Section 3 we present the methodology for evaluating BPMS. Then, in Section 4 we present case studies regarding the evaluation of open source and proprietary BPMS. Finally, in Section 5 we present some conclusions and future work.

2 Related Work

There are several approaches for evaluating BPMS, which we have analyzed and taken into account when defining our methodology and the list of characteristics we provide. We have taken into account software quality characteristics standards such as ISO/IEC 9126 (superseded by SQUARE [11]), and others such as [12]. Although the characteristics defined in these are not specific to BPMS, some can be applied to software of any kind and are very important for evaluating the quality of software from different points of view. A key reference from academy for evaluating BPs languages capacities and BPMS support for modeling and execution is the workflow patterns [7], used for example in [13] to evaluate the support provided by selected open source tools. This kind of assessments can identify potential limitations for modeling and execution of BPs in the selected languages and/or BPMS that is better knowing in advance. Other works such as [8] evaluate selected open source workflow engines against their compliant to the WfMC model, defining key characteristics for them. Other evaluations are contemporary to our work, such as [14] in which WS-BPEL engines; both open source and proprietary are evaluated. They differ from ours in many aspects a key one is that ours is more generic and allows many types of engines to be evaluated and compared, since it is not restricted to a single language. Other proposals are more generic for the selection of any type of COTS software such

[1] http://www.fing.edu.uy/inco/investigacion/grupos/COAL

as [15,16] or with focus on specific characteristics of OSS software such as [17]. In [18] the authors present a survey on processes for selecting COTS. Although our proposal shares some similarities with these works regarding the process and some general desirable characteristics, our focus is on BPMS for which our list of characteristics provides a unique insight of this type of systems. Moreover, we provide several test cases and guides to evaluate the characteristics we defined.

Regarding industry reports of evaluation of proprietary BPMS, we have mainly considered Gartner [9] and TEC [10] reports, and also checked out the Forrester [19] approach. Gartner evaluations we have analyzed include the Magic Quadrant for BPMS [20] and Magic Quadrant for iBPMS [21], the latter adding elements of Business Intelligence. The criteria used by Gartner include commercial characteristics, such as: price, customer experience, market understanding and strategy, business model, among others. TEC provides software features for evaluating different types of software, in a web application with a list of defined and categorized characteristics, obtaining a recommendation of selected tools that best suits the indications provided. Forrester [19] also provides a similar approach with a set of characteristics and a tool to support the recommendation. However, these approaches are based on information provided by vendors who answer a questionnaire valuating each characteristic (Forrester also includes laboratory evaluation). In [20,21] the results are not context-dependent since the importance of the characteristics is given and not selected for each evaluation. Unlike these works, our approach does not include any view from the vendors themselves, but from our objective evaluation from carrying out each test case for each defined characteristic on each selected BPMS. The importance of the characteristics is also assigned each time by each organization, guaranteeing that the results are adequate for each organization every time.

3 BPMS Evaluation Approach

The main results of our work are the list of characteristics which can be used as a basis for evaluating BPMS, and the methodology to carry out the evaluation.

3.1 List of Characteristics

The list is organized according to a defined structure which groups characteristics allowing an intuitive understanding. The highest level corresponds to the modules, which in turn are composed of categories grouping cohesive characteristics. We have analyzed and selected characteristics based on many sources, as presented in Section 2. Two main modules are defined: (1) Technical, which involves everything related to software itself, and (2) Non-technical, which encompasses other characteristics such as community support. Table 1 shows the defined structure including both modules and its categories, the total of characteristics defined within each one (# DC) and, due to space limitations, an example of characteristics in each category.

Table 1. Structure and example of defined modules, categories and characteristics

Module	Category	# DC	Example of characteristics
Technical	Technology, Architecture and Interoperability	15	• BPMS Architecture • Integration with social networks
	Process Design and Modeling	12	• Modeler type • List of versions of process models • Collaborative work on processes
	Form management	9	• Dynamic Forms • Support for mobile devices
	Workflow engine	24	• Support of workflow patterns • Configurable Schedule of full System • Linking task and document
	Security Management	5	• Permission Mechanism for users • Role Definition
	Management, Monitoring and Audit	9	• Process Monitoring • Backups
	Document management system	5	• Integrity and document security • Indexing and search mechanisms
	Portal	7	• Customizing the portal • Searching Mechanisms
Sub-Total		79	
Non-Technical	Installation and support	8	• Installation packages • Available documentation • Supported Languages
	Maturity	6	• Time in market • Community activity
	Commercial	1	• License costs
Sub-Total		15	
Total		94	

3.2 Evaluation Methodology

Fig. 1 models the proposed evaluation methodology using BPMN, showing the different activities to be carried out within each organization, including the sub-process of actually evaluating the tools. In the first place the list of characteristics is reviewed and updated if needed and the tools to be evaluated in the current evaluation are selected, based on initial criteria such as being open source or proprietary, the language provided, among others defined to help narrow the selection. Then, each characteristic is weighted by the organization using a scale we provide, both to define how important is each characteristic to the organization and to obtain a ranked list to select the most important characteristics to be evaluated (as evaluating all of them can be expensive and time consuming). Then the test cases to evaluate the selected characteristics are defined (or adapted if needed, as we provide many test cases) and the case study to be carried out within each tool is specified (as we also provide many case studies). Then, the evaluation is performed valuating each characteristic within each tool in another scale we provide for results (using the test cases for each one and the case study to assess the tool globally). Finally a total score for each tool is calculated.

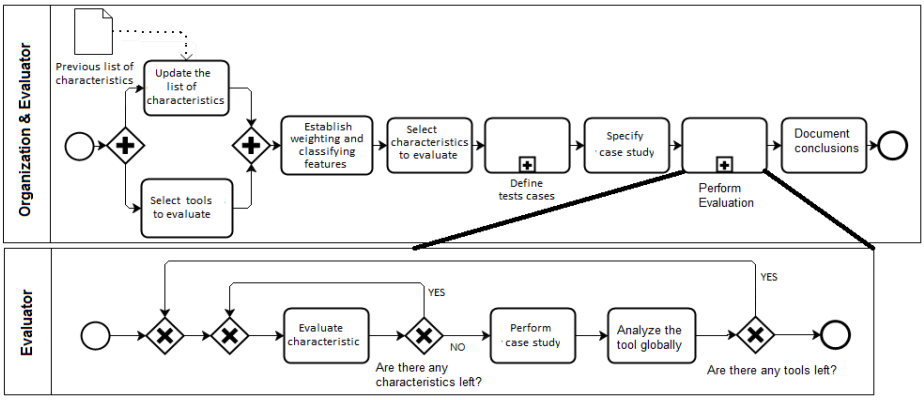

Fig. 1. Evaluation methodology process modeled in BPMN

The list of defined characteristics for BPMS tools is reviewed and updated for each evaluation, when needed. This update could involve the addition of new characteristics, modifying existing ones and/or deleting other, allowing working with a suitable list of characteristics at the time the evaluation is made. In parallel with this, the tools to be evaluated are selected (from those lists provided by organizations such as OMG, OASIS and WfMC), based on initial criteria defined such as open source or proprietary, presence in a specific market, or support for a defined modeling notation.

Each characteristic is then assigned a level of importance by the organization performing the evaluation. The scale defines the following levels: (1) Mandatory; (2) Medium priority and (3) Low priority. The classification of the characteristics on this scale depends on the needs of the organization for each evaluation and therefore allows to instantiate the evaluation to the organizational context. Each characteristic is further valuated in a three level scale of support: (1) Totally supported, the tool has the characteristic; (2) Partially supported, the tool does not cover the entire specification of the characteristic; (3) Not supported, the tool does not provide it. Additionally three levels of compliance are defined for the support scale: (1) Native, the feature is part of the tool; (2) Particularization, specific software can be developed to achieve such compliance; (3) Integration, it is necessary to include a third component to support it. In order to obtain a quantitative evaluation (score) associated with each tool, we also calculate a final value regarding the importance defined and the results level. Moreover, when two tools present different levels with respect to the same support scale, e.g. third components are needed but with different development costs, we can assign different values. The list of ranked characteristics regarding their importance as assigned by the organization is reviewed to select the key ones to be used in the current evaluation, to help reduce the number of characteristics to be evaluated. Nevertheless all the characteristics could be selected if the organization wants, taking into account that it will require more time and effort. Since the list of characteristics provide a shared criterion for evaluating BPMS, previous results of evaluations can be used as a basis for carrying out an organization-dependent evaluation process.

Two ways for evaluating the characteristics are defined: theoretical and practical. The theoretical evaluation does not require executing the tool, but is mainly based

on the tool documentation, e.g. when non full versions are available or when characteristics are not a priority for the organization. The practical evaluation do requires executing the tool, with a specific test case to evaluate the level of support it provides. The main purpose of the test cases is to standardize the evaluation with respect to the same basis. In addition, a case study is defined to be performed within all tools. The main objective of the case study is to give an overview of the support provided by the tool regarding the actual execution in a daily basis operation.

Finally a global analysis of each tool is performed, including the score assigned by the defined formula. This allows performing a comparison between tools based on each characteristic evaluation result and the overall score assigned to each tool.

4 BPMS Evaluation Case Studies

In this section we present the results of evaluation projects of open source and proprietary BPMS we have carried out between years 2010 and 2013 as a validation and assessment of our proposal. We have followed the guidelines in [22] for cases studies, but due to space limitations we present here only the results discussion.

4.1 Open Source BPMS Evaluation

The open source BPMS assessment was carried out in two projects, one focusing on BPMS based on the XPDL standard and another on the WS-BPEL standard. Two more tools were also selected which implemented the new BPMN 2.0 standard which was released in the course of the evaluation projects. The following tools were selected and evaluated mainly based on their availability:

- XPDL: Bonita CE[2], Enhydra[3], Joget[4], OBE[5], WfMOpen[6]
- WS-BPEL: Apache ODE[7], jBPM[8], Orchestra[9], Petals[10], Intalio CE[11], Riftsaw[12]
- BPMN 2.0: Activiti[13], jBPM5[8]

Fig. 2 shows an example of the results for the evaluation of some selected characteristics for XPDL and BPMN 2.0 (Activiti) and WS-BPEL and BPMN 2.0 (jBPM5) engines, using the semaphore metaphor: Green for Totally supported, Yellow for Partially supported and Red to indicate Not supported.

---

[2] BonitaSoft BPMS, http://www.bonitasoft.com/
[3] Together Enhydra Shark, http://www.together.at/prod/workflow
[4] Joget, http://www.joget.org/
[5] OBE, Open Business Engine, http://obe.sourceforge.net/
[6] WfmOpen, http://wfmopen.sourceforge.net/
[7] Apache ODE, http://ode.apache.org/
[8] jBPM, jBPM5, http://www.jboss.org/jbpm/
[9] Orchestra, http://orchestra.ow2.org/
[10] Petals, http://petals.ow2.org/
[11] Intalio BPMS, http://www.intalio.com/bpms
[12] Riftsaw, http://www.jboss.org/riftsaw
[13] Activiti BPMS, http://www.activiti.org

(a) XPDL and BPMN 2.0	Enhydra	Joget	Bonita	OBE	WFMOpen	Activiti
Flow routing	🟡	🟢	🟡	🟢	🟡	🟢
Process Designer	🟢	🟡	🟢	🔴	🔴	🟢
Calendar	🔴	🔴	🔴	🔴	🔴	🔴
Business Rules	🟢	🟢	🟢	🟢	🟢	🟢
Notifications and Alerts	🟡	🟡	🟢	🟢	🟡	🟢
Tasks Allocation by Roles	🟢	🟢	🟢	🔴	🟡	🟢
Work lists	🟢	🟢	🟢	🟢	🟡	🟢
Task assignments	🟡	🟡	🟢	🔴	🟡	🟡
Process and/or Activity Monitoring	🟢	🟢	🟢	🟢	🟡	🟡

(b) WS-BPEL and BPMN 2.0	jBPM BPEL	Apache ODE	Intalio	Riftsaw	Petals	Orchestra	jBPM5
Easy of installation	🟡	🟡	🟢	🟡	🟢	🟡	🟢
Deploy assistance	🟢	🟢	🟢	🔴	🟢	🟢	🟢
Execution of multiple versions of a process	🟢	🟢	🟢	🟡	🔴	🟡	🟡
Suspend or rollback a process	🔴	🔴	🟡	🔴	🟡	🟡	🔴
Audit	🟡	🔴	🟢	🟡	🔴	🔴	🔴
Reports	🔴	🔴	🔴	🔴	🟡	🔴	🟢
Integration with different Data Bases	🟢	🟢	🟢	🟢	🟢	🟢	🟢
Obtaining execution indicators	🟢	🟡	🟢	🟡	🟡	🔴	🟡

Fig. 2. Example of evaluation results: (a) XPDL and BPMN 2.0; (b) WS-BPEL and BPMN 2.0

Fig. 3 shows the overall scores obtained by each tool in each evaluation, by means of the formula defined (c.f. Section 3.2). As mentioned above, to obtain those values the importance assigned to each characteristic by each organization performing the evaluation is also taken into account, to weight the most important ones. This final score serves mainly for the purpose of "eliminating" tools which does not reach certain level, focusing on the ones presenting the best scores.

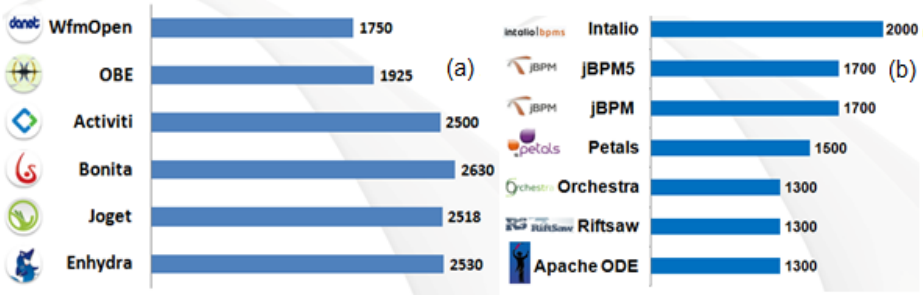

Fig. 3. Overall scores for evaluations: a) XPLD and BPMN 2.0; (b) WS-BPEL and BPMN 2.0

As conclusions some interesting observations were found: in the first place the version of the XPDL standard of the engine plays an important role in their capabilities, as it has several versions, 1.0, 2.0 and 2.1 with different features. Four of the six engines implemented version 1.0 which has significant limitations. Versions 2.0 and 2.1 already have elements of the BPMN standard and therefore better capabilities. Some of the engines had a high complexity of installation at the time, as WfMOpen and OBE. Bonita presents support for most features, including process simulation, being one of the most complete engines. Enhydra and Joget for XPDL and Activiti for BPMN 2.0 also provide support for most of the characteristics evaluated. Being the Activiti an initial version of the BPMN 2.0 standard implementation, it was expected to improve considerably, which has already occurred in recent years. As for the

execution language WS-BPEL all evaluated engines implement the WS-BPEL 2.0 standard and some like Intalio CE, include extensions to provide support for humans such as HumanTask or BPEL4People. Intalio CE is the engine that provides better support for the evaluated characteristics and better results throughout the evaluation, with a friendly environment, stable behavior and an active community. Other engines such as jBPM, jBPM5 and Petals, also provide support for most characteristics.

4.2 Proprietary BPMS Evaluation

The evaluation on proprietary tools also included a reevaluation of the Bonita open source BPMS since a major update was performed in the newly released 6.0 version. The selection was mainly based on their presence in the local market, and included:

- Proprietary: IBM BP Manager [14], Oracle BPM[15], Apia[16], GXFlow[17]
- Community edition: Aris Platform[18], Bizagi[19], Bonita[1].

Fig. 4 (a) shows an example of the results obtained from evaluating selected characteristics on the tools. As before, the semaphore metaphor is used, and the theoretical and practical evaluations are shown. Being proprietary tools, for the practical evaluation it was necessary to obtain licenses for the products to evaluate. Due to difficulties in obtaining the software and/or corresponding licenses, two tools were only evaluated theoretically, while the rest were also evaluated in practical (cf. Section 3.2), as shown in Fig 4 (a). The overall score is shown in Fig. 4 (b) which is obtained by applying the formula as before, taking into account both the importance defined for each characteristic by the organization and the results obtained by the characteristic evaluation. We do not disclose which value corresponds to which tool for confidentiality reasons regarding definitions in the projects carried out with the enterprises involved.

As conclusions of this evaluation we can highlight as a key point the original nature of the tool. Three of the tools have as main objective support for BPM, so their architecture and feature set are intended for these purposes; while the remaining tools are extensions of larger tools which were developed with other objectives. Those tools which focus on BPM have advantages in characteristics such as: installation, usability, understanding, documentation, simpler architectures, etc. The other tools must adapt to certain preset parameters, providing less specific support to BPM, increasing the learning curve and presenting more installation problems, among others. However, the latter proved more powerful, with more features available, management tools and administrative consoles. As for the language of the process engine both XPDL and WS-BPEL are provided for process execution, and there is less support for BPMN 2.0 which is mainly provided within modelers but not for execution. In most of the tools wizards support is provided for BPs implementation helping reduce the development time, for example to integrate web services from WSDL definitions.

---

[14] IBM BP Manager, http://www-01.ibm.com/software/ integration/business-process-manager/
[15] Oracle 11g, http://www.oracle.com/us/technologies/bpm/suite/overview/
[16] Apia, http://www.statum.biz/web/guest/apia
[17] GXflow, http://www.genexus.com/gxflow
[18] ARIS Platform, http://www.softwareag.com/corporate/products/aris
[19] Bizagi BPM Suite, http://www.bizagi.com/

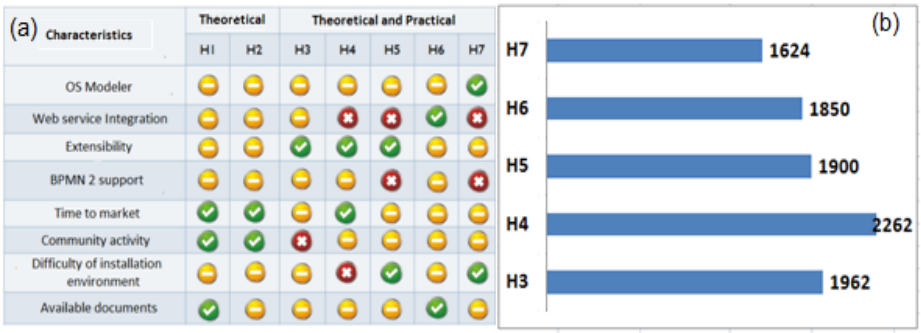

Fig. 4. Sample of (a) characteristics evaluated in Proprietary tools (b) overall scores

5 Conclusions

We have presented a systematic approach for evaluating BPMS tools both open source and proprietary, which includes a methodology and a list of key characteristics for this kind of software, as well as a way to instantiate each evaluation to the specific context of the organization performing it, providing different results for different needs. We provide a list of relevant key characteristics for BPMS tools, and a way of evaluating the provided support by means of tests cases and a case study to provide an overall view of the tool support. We believe that the evaluation methodology we propose can be also applied to other type of software, by changing the list of characteristics according to the software being evaluated, and following the defined process. We have also illustrated the use of the approach by means of case studies regarding open source and proprietary BPMS. We can conclude that all tools have advantages and disadvantages, and can be suitable for different contexts. As there is an increasing demand from organizations to incorporate BPMS platforms, we believe that a key element and contribution of our evaluation approach is to take into account the organizational context, allowing different organizations to select different tools regarding their specific needs. We can also conclude that regarding this kind of software some open source BPMS such as Bonita, Activiti and Intalio CE are, with respect to some aspects, as complete and competitive as other products from major existing vendors.

As future work, we plan to evaluate again some open source BPMS since they all have improved significantly in the last few years (we have continued using them in many projects and courses, mainly Activiti and Bonita), and to add new others. We also plan to provide tool support for the methodology by generating a benchmark for the evaluation as well as a tool allowing evaluators and users to assign weights to the characteristics and generate recommendations and comparisons between tools. Finally, we plan to include other aspects to the evaluation, e.g. non functional aspects.

Acknowledgements. We would like to thank the students who worked in the BPMS evaluation projects: A. Acosta, N. Beloso, R. Bonini, E. Galindo, A. Hernández, F. Parins and C. Smith.

References

[1] Weske, M.: BPM: Concepts, Languages, Architectures. Springer (2007)
[2] van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M.: Business process management: A survey. In: van der Aalst, W.M.P., ter Hofstede, A.H.M., Weske, M. (eds.) BPM 2003. LNCS, vol. 2678, pp. 1–12. Springer, Heidelberg (2003)
[3] Chang, J.F.: BPM Systems: strategy and implementation. Auerbach Publications, Taylor & Francis Group (2006)
[4] OMG. Business Process Model And Notation (BPMN) v2.0 (2011)
[5] WfMC, XML Process Definition Language (XPDL) (2008)
[6] OASIS, WS Business Process Execution Language (WS-BPEL) (2007)
[7] van der Aalst, W.M.P., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Dist. & Parallel Databases 14(3) (2003)
[8] Garcês, R., Jesus, T., Cardoso, J., Valente, P.: Open source workflow management systems: A concise survey. In: 2009 BPM & Workflow Handbook, Future Strategies Inc., pp. 179-190 (2009)
[9] Gartner Group. http://www.gartner.com/technology
[10] TEC, Technology Eval. http://www.technologyevaluation.com/
[11] ISO/IEC 9126, Software engineering, Product quality, superseded by ISO/IEC 25000 SQuaRE. http://www.iso.org/
[12] Tsalgatidou, A.: Selection criteria for tools supporting business process transformation for electronic commerce. In: Proceedings of EURO-MED NET (1998)
[13] Wohed, P., Andersson, B., ter Hofstede, A., Russell, N.C., van der Aalst, W.M.P.: Patterns-based Evaluation of Open Source BPM Systems: The Cases of jBPM, OpenWFE, and Enhydra Shark. Inf. Softw. Technol. **51**(8), 1187–1216 (2009)
[14] Harrer, S., Lenhard, J., Wirtz, G.: Open source versus proprietary software in service-orientation: The case of BPEL engines. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 99–113. Springer, Heidelberg (2013)
[15] Morera, D: COTS evaluation using desmet methodology & analytic hierarchy process (AHP). In: Oivo, M., Komi-Sirviö, S. (eds.): PROFES 2002. LNCS vol. 2559, pp. 485-493. Springer, Heidelberg (2002)
[16] Lawlis, P., Mark, K., Thomas, D., Courtheyn, T.: A Formal Process for Evaluating COTS Software Products. IEEE Comput. **34**(5), 58–63 (2001)
[17] Taibi, D., Lavazza, L., Morasca, S.: OpenBQR: a framework for the assessment of OSS. In: OSS 2007, pp. 173-186 (2007)
[18] Tarawneh, F., Baharom, F., Yahaya, J., Ahmad, F.: Evaluation and Selection COTS Software Process: The State of the Art. International Journal on New Computer Architectures and Their Applications (IJNCAA), 344-357
[19] Richardson, C., Miers, D., Cullen, A., Keenan, J.: The Forrester Wave: BPM Suites, Q1 2013, How The Top 10 Vendors Stack Up For Next-Generation BPM Suites, March 2013
[20] Sinur, J., Hill, J.: Magic Quadrant for Business Process Management Suites. Gartner Inc. (2010)
[21] Sinur, J., Schulte, W., Hill, J., Jones, T.: Magic Quadrant for Intelligent Business Process Management Suites. Gartner Inc. (2012)
[22] Yin, R.: Case Study Research: Design and Methods. Sage Publications, Inc. (2002)

Smart Route Planning Using Open Data and Participatory Sensing

Vivek Nallur(✉), Amal Elgammal, and Siobhán Clarke

FutureCities, Distributed Systems Group, Trinity College, Dublin, Ireland
{vivek.nallur,amal.elgammal,siobhan.clarke}@scss.tcd.ie

Abstract. Smart cities are not merely the infusion of technology into a city's infrastructure, but also require citizens interacting with their urban environment in a smart and informed manner. Transportation is key aspect of smart cities. In this paper, we present a smart route planning open-source system; SMART-GH utilizes open data and participatory sensing, where citizens actively participate in collecting data about the city in their daily environment, e.g., noise, air pollution, etc. SMART-GH then augments the routing logic with sensor data to answer queries such as 'return the least noisy route'. SMART-GH enables citizens to make smarter decisions about their daily commute, and subsequently improve their quality of life.

Keywords: Participatory sensing · Open-data · Open-source · Smart-city-routing

1 Introduction

There are a plethora of route-planning applications available to citizens today. Web-based applications (*e.g.*, Google Maps, Live Maps, Yahoo! Maps, etc.), specialized device-based applications (*e.g.*, TomTom, Garmin, etc.) and smartphone-based apps (*e.g.*, iOS Maps, Google Maps, etc.), all allow a user to plan a route from one point to another, calculating the fastest possible route, or shortest route. There are some that even offer to calculate the fastest route using 'current' traffic data. However, all of these applications rely on a centralized source of data, usually controlled by a single entity.

As cities get smarter, and more sensors are available in the urban environment, route planning can be augmented to use this sensor data. For example, elderly citizens, parents with young children, etc. would like to know which streets have the least air pollution or least noise, along their route? Currently, there are two roadblocks to answering such questions: (i) how do we get sensor

A. Elgammal—Assistant Professor, Faculty of Computers and Information, Cairo University.

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 91–100, 2015.
DOI: 10.1007/978-3-319-17837-0_9

data about city-scale locations into the hands of the citizens? and (ii) how can this aggregated data be utilized by individuals?

As an answer, this paper presents a case-study of an application that combines *participatory sensing* [4], *open-data* from government sources, and integrates it with open-source routing libraries to provide a *smarter* route. An application that provides a social good, if created without participatory sensing, would require considerable investment in a sensor network from either the government, or an enterprise. Even if this were feasible, smart-city applications due to their pervasive nature, raise issues of privacy [3] from government databases. Not only does open-source software and open-data help in the provision of a social good, but it also alleviates *some* of the privacy issues that are inevitable, in a smart-city application. The rest of this paper is organized as follows: Section 2 introduces concepts of aggregating city-level data, while Section 3 presents the structure and behaviour of our smart-city application. Discussion and future work are addressed in Section 4, followed by concluding notes in Section 5.

2 Participatory Sensing, Open Data and Smart Cities

2.1 Opportunistic/Human-Centred/Urban/Participatory Sensing

Traditionally, wireless sensor networks have been used by researchers and city administrators to collect data about specific aspects of, and specific locations, in a city. This method has the obvious disadvantages of: (i) being expensive to deploy, (ii) high maintenance cost due to field-damage, and (iii) requiring owner permission for installation

Given the penetration of smartphones [9], and the possibility of using smartphones as sensors, at least two of the three disadvantages above can be eliminated. Ethically, the third disadvantage continues to persist, but can be alleviated through education and transparency. Citizens must be motivated to donate data to the city, and thereby help create a city-level aggregation of the desired data. However, most citizens are only motivated to participate, if they perceive a benefit to contributing their time and data [4]. We posit that such sensor data, made available by individuals, can be leveraged easily and harnessed to provide a social good. For example, individuals with breathing difficulties can use localized pollution data to determine how to plan their commute to work. In this paper, we describe a case-study of a novel application that leverages open-data contributed by individuals, and allows them to reap the benefits of such contributions by enabling smart route planning, which incorporates participatory as well as open data sources into the routing algorithm.

2.2 Open Data

Many public institutions and governments release data being generated from their research and data gathering bodies. The EU releases its data from the EU Open Data Portal[1]. Ireland does not yet have a centralized portal that covers

[1] https://open-data.europa.eu/en/data/

all departments, but it has announced a national action plan [2] that lays out a roadmap for open-data to be released. Most open-data available in Ireland is through city councils' websites. Dublin city council releases data available on Dublin city through its *Dublinked* initiative[3]. Use of *open-data* is expected to provide macro-level insights into various aspects of a government, and a city's life. In particular, from this paper's perspective, it allows us to leverage existing environmental data that is available about Dublin City, and incorporate it into routing decisions. **Note:** the use of Dublin city in this paper is merely illustrative, and the system that will be described is in no way tied to any particular city.

2.3 Smart Cities

The term *smart city* is frequently used to connote the use of information technology by city authorities, in various domains ranging from smart-grids to waste-management to intelligent-transport, etc. to enable a better quality of life for citizens. The basic idea is that cities could (and should) actively leverage the latest technologies, to make informed decisions at the macro-level, and enable citizens to make intelligent individual choices about city resources. About 180,000 people move to cities everyday and it is predicted that by 2050, about 70% of the world's population will be living in urban agglomerations [8]. This will result in huge pressure on resources such as energy, water as well as exacerbate problems of congestion, waste-management, etc. In Dublin alone, the cost of congestion is estimated at 4.1% of the GDP [6]. In such a scenario, judicious and smart use of technology to ease problems in transport, healthcare, energy-usage and waste-management, etc. is of utmost importance. However, current technical solutions cannot simply be transplanted into a city, typically due to the following problems:

- **Multi-ownership:** Different parts of a city's infrastructure are owned by different agencies, and hence technical solutions developed by one are not necessarily interoperable with others. Also, different agencies have different goals, and it is difficult to create a system that can account for all possible goals
- **Scale:** Most decision-making mechanisms do not scale to city-scale entities, when information and goals are effectively de-centralized
- **Availability of fine-grained and yet aggregate data:** To be able to take decisions at a city-scale, fine-grained and up-to-date data needs to be made available on a continuous basis. This is currently difficult to achieve. Either data is not available, or decision-makers are flooded with data, but do not have a macro-level perspective.

In this paper, we focus on a smart-city application that circumvents the data availability issue. This application assumes that there will be no single data stream, allows individuals to contribute data, and then uses the aggregated data to provide a service back to individual citizens.

[2] http://www.per.gov.ie/minister-brendan-howlin-td-publishes-irelands-first-open-government-partnership-national-action-plan/

[3] http://www.dublinked.ie/

2.4 Issues

Smart-City applications tend to use technologies that are pervasive in nature. That is, sensors and sensor-networks that surround the user throughout the day. A smartphone is a perfect example of a sensor-based device that is both, uniquely identifiable as well as pervasively present. A smart-city application that uses participatory sensing is uniquely positioned to identify individuals and their habits, in ways that the user did not anticipate. In such a situation, it is imperative that the application make extra efforts to ensure that user-privacy is protected.

While governments are sometimes willing to open up data sources, there have been criticisms [7] about their intent in doing so. A major critique is the utility of releasing raw data, when the citizen is unable to interpret such data and make informed decisions. Helbig *et al.* report that most government websites are essentially data dumps with little thought given to usability, quality of data, or consequences of use [5]. In such cases, open-data merely becomes a way for public authorities to pay lip-service to openness, without actually achieving anything concrete.

3 Smart GraphHopper

Our smart-city application called Smart GraphHopper[4] (*Smart-GH* for short) uses GraphHopper[5], which is an open-source routing library that uses OpenStreetMap (OSM). The original GraphHopper can be used to plan either the *fastest*, or *shortest* routes[6] in the same way as its commercial counterparts (GoogleMaps, etc.). GraphHopper uses the *length* and *maxspeed* data embedded inside the OSM to calculate routes. However, Smart-GH extends GraphHopper to allow citizens to compare routes by evaluating different available sensor data, e.g., noise data, air pollution data, etc. It also enables better distributed deployment capabilities.

Participatory Sensing: A smart-city application relies on city-wide data. Participatory data could be a key enabler in providing the application with a large set of data points. For this purpose, we use NoiseTube [2], which is a mobile app, developed by *Vrije Universiteit Brussel*. NoiseTube uses sensors on smartphones and measures the ambient noise in the user's surroundings.

Open Data: Air quality monitoring data is available from Dublin City Council's *Dublinked* portal[7]. The city council maintains fixed sensor stations covering the major areas of Dublin city (centre, north, south, east, west).

In the next sub-sections, the structure, behaviour and technological choices to build and deploy Smart-GH are discussed in more detail.

[4] https://github.com/DIVERSIFY-project/SMART-GH

[5] https://github.com/graphhopper/graphhopper/

[6] In this paper, we refer to the original GraphHopper as *GraphHopper*, and to our extensions to transform it into a smart and distributed GIS system as *Smart-GH*.

[7] http://dublinked.com/datastore/datasets/dataset-279.php

3.1 Processing Data

Smart-GH supports both volunteer data as well as open data. The three essential elements required for *any sensor data* to be integrated into Smart-GH are: (i) *location* in the form of GPS coordinates, (ii) *value* read by the sensor and (iii) *timestamp*. Smart-GH requires each city to have a `config` file, that lists all the sensor-types that are available. In this case, Dublin city has two sensor types: (a) Noise (b) Air Pollution.

Noise data is stored on mobile phones as XML files that can be automatically uploaded to the server. Researchers in Trinity College Dublin (TCD) installed NoiseTube on their smart phones and started collecting noise data during their daily commute. Using NoiseTube's *API*, Smart-GH periodically pulls data about noise levels on Dublin's streets.

Air pollution data from Dublin City Council is available as a set of excel sheets, which were converted into `csv` format. Smart-GH currently supports three file formats: `xml`, `json`, and `csv`. These file reading mechanisms are inserted via a plugin mechanism, and can be extended to handle any other file-format, as desired.

Parsing and Filtering: For each sensor-type, the `config` file lists the name of a datasource, parser and a filter. The Parser is responsible for connecting to the datasource (web service returning json, or csv files), and collecting all the relevant data. The Filter is used to prevent fluctuations in the day-to-day collection of sensor data, from affecting the final value. The currently used filter is exponential weighted moving average, however any other filtering mechanism may also be simply plugged in, via a config file. There is a default no-op filter.

Reverse-Geocoding: Typically, each sensor reading is linked to a GPS location. To link them to the relevant OSM map ways (e.g. streets, roads, etc.), the Parser needs to obtain concrete way-ids for each GPS location in the respective files, a process known as *Reverse-Geocoding* [1]. For this, the Parser calls a geocoding web service[8], which takes a GPS-coordinate as input and returns concrete OSM way-ids.

Data Storage: The OSM information is now associated with the sensor value, and stored as a hash in Redis. This is done for both, the noise as well as the air pollution sensor. However, collecting the air pollution sensor data is less computationally intensive, since the sensor locations are fixed and *Reverse-Geocoding* becomes a one-off process.

Figure 1 presents a UML sequence diagram capturing the behaviour of the actors involved and the flow of logic. As shown in the figure, there are six system actors: 'Data Retrieval and Processing Daemon', 'Air Pollution Sensor', 'Parser', 'Filter', 'Reverse geocoding Web service' and 'Redis', and one human actor: 'Noisetube Participator'. The scenario is started by the 'Data Retrieval and Processing Daemon' getting the noise and air pollution readings from 'Noisetube

[8] http://services.gisgraphy.com/street/streetsearch

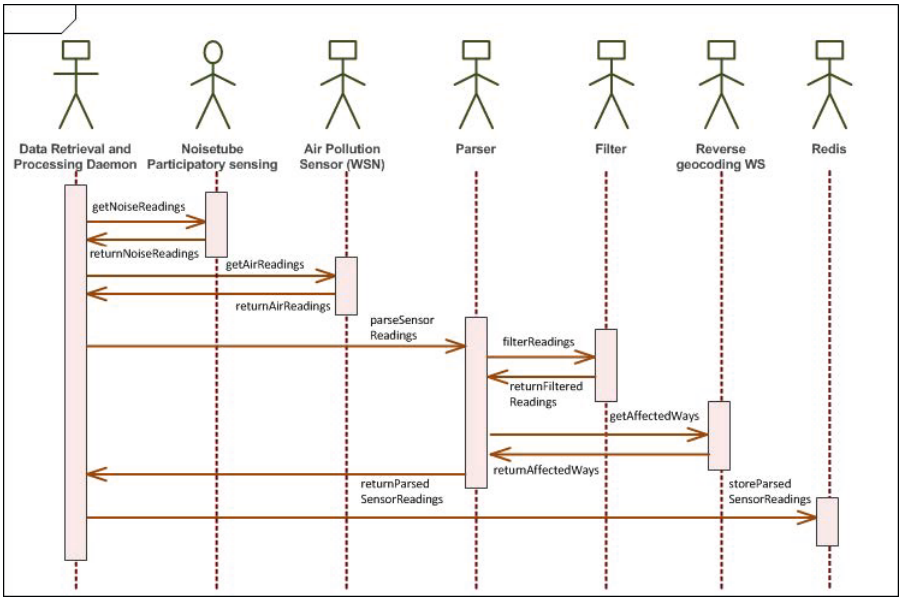

Fig. 1. UML sequence diagram of sensor data reading and parsing

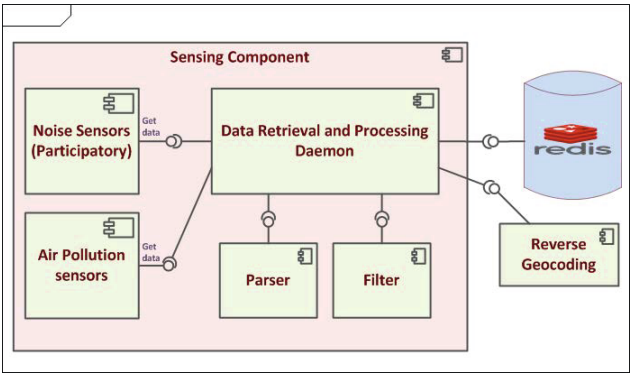

Fig. 2. UML component diagram of Smart-GH sensor data reading and parsing

Participatory Sensing' and 'Air Pollution Sensor', respectively. Figure 2 presents a finer-grained UML component diagram of this sensing component.

3.2 Marrying the Sensors to Routing

As discussed above, Smart-GH extends the typical usage of GIS applications by integrating sensor data from various sources, which will then be used as the basis of routing decisions. Smart-GH utilizes the GraphHopper routing engine for route planning. GraphHopper is a fast, efficient routing library that

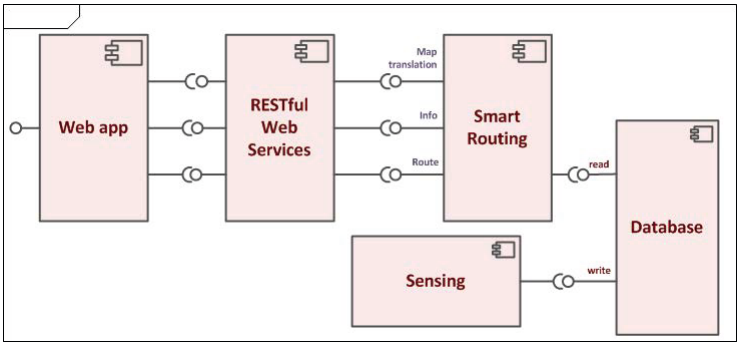

Fig. 3. UML component diagram of Smart-GH

implements various routing algorithms, which given two (up to five) GPS locations, can calculate either the *fastest* or the *shortest* route, connecting these points. The fastest/shortest route is calculated based on speed limits/distance tags associated with OpenStreetMaps (OSM) ways (streets, roads). The problem of finding a fastest or shortest route becomes a minimization problem: minimum total time or minimum distance, respectively.

Enabling routing based on collected sensor readings required extension of the original GraphHopper library. First, the communication protocol was modified to allow arbitrary sensor types to be included as the user's preferred routing mechanism. In this case, 'noise' and 'air-pollution' are two additional values that are available to the user. Second, we modified the existing algorithms to perform routing based on stored sensor readings in Redis, instead of OSM tags. Figure 3 shows a UML component diagram of Smart-GH after augmenting it to enable smart routing. As shown in the figure, Smart-GH constitutes four components: 'Sensing', 'Database', 'Routing' and 'Web app'. The 'Sensing' and 'Database' represent the data sensing and parsing component and Redis database, as explained previously in Section 3.1.

Figure 4 presents a screenshot of the Smart-GH web interface, which visualizes a Bike route from Smithfield to School street(two places in Dublin), where the weighting is selected as 'Least Noisy', by dynamically visualizing noise data as a heatmap overlay. GraphHopper has a very simple (and limited) web interface that allows a user to only query for the fastest driving (car) route between two GPS locations. An *Android App*[9] has also been developed for SMART-GH. Both apps have been designed and developed to meet the following functionalities:

– Enable the user to request sensor-based weights on routes. Current options for Dublin city are: fastest, shortest, least noisy, least air pollution. We maintain a city configuration file for each city, capturing, among others, the types of sensors available for this city. Based on available sensor data for each city, the web-interface is automatically configured to include the weighting specified in the relevant city config file. The appropriate config file is parsed

[9] https://github.com/DIVERSIFY-project/SMART-GH/blob/master/SMART-GH-Android/platforms/android/ant-build/CordovaApp-debug.apk

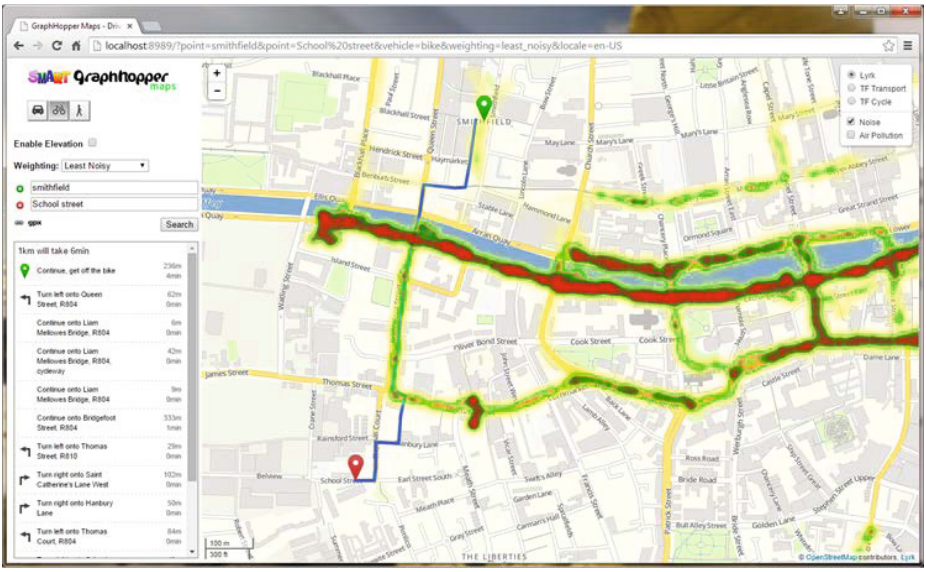

Fig. 4. Screenshot of Smart-GH web interface

based on the loaded map. For example, since `dublin.osm` map is loaded in Figure 4, `dublin.config` is parsed accordingly. Since `dublin.config` has two sensors corresponding to noise and air pollution data, the interface is amended to the drop-down list as shown in Figure 4.

– Support more than one vehicle type on the same running session. As shown in Figure 4, car, bike and foot are supported. These configuration parameters are specified in Smart-GH config file.

We modified the interaction model between the components to be *asynchronous* and *stateless*, and then wrapped the routing algorithms into Restful Web Services (WS). This allows the system to be scaled horizontally, to meet increased (web)traffic.

4 Discussion and Future Work

4.1 Performance

The deployment configuration shown in Figure 5 shows the sensor datastore on a separate machine, with Redis serving up the data. One of GraphHopper's biggest strengths is *speed*. Due to its unique caching strategy, it is able to provide routes over long distances ($> 100km$) very fast. Accessing sensor data, is an out-of-process procedure, and is therefore much slower than if only OSM tags are used. However, in a smart-city usage scenario, route demand is rarely greater than 100km, while access to sensor data provides much more value. Also, due to data-retention/privacy laws, it might not be possible to always hold sensor

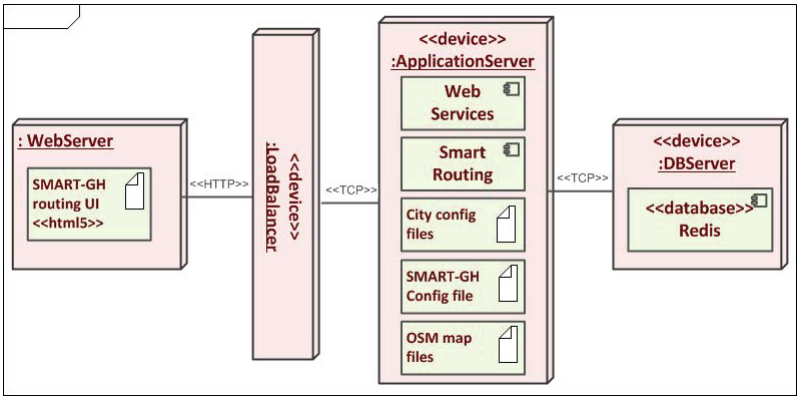

Fig. 5. UML deployment diagram of the distributed deployment of Smart-GH via WSs

data on Smart-GH's servers. We believe that flexibility afforded by an out-of-process access to a datastore is an acceptable trade-off to the performance-loss of in-process routing.

4.2 Security/Privacy

Participatory Sensing, in some ways, depends on the *kindness of strangers* to be effective. To acquire a critical mass of data and ensure continuous updates, it is essential to gain the trust of citizens by ensuring data anonymity and security. Even with an innocuous application like NoiseTube that makes no conscious effort to uniquely identify its users, it is fairly easy to correlate *usernames* (from the data collected) to the routes they have taken, and thus find out where they work, and reside. In the data-processing performed by Smart-GH, we strip away **all** user information, and work **only** with street locations and noise-levels. For a smart-city application, privacy is an aspect that must be explicitly and aggressively engineered in, since the default protocol of releasing open-data does not ensure this. Given that our entire software chain is made of open-source components, it is easy to verify that the data is anonymized, as early as possible.

4.3 Sensor-X

Currently, Smart-GH knows all the sensor-types that are present in a city. Although, for a new sensor-type, a new parser and filter can simply be plugged in through the `config` file, the routing algorithm still needs to know which sensor data it is dealing with. In the ideal case, users must be able to extend Smart-GH to handle different sensors without having to modify the routing algorithm. This will allow different subsets of users in a city to add different kinds of sensors (and corresponding data) without having to contact Smart-GH developers. In other words, the application should be *self-extending*.

5 Conclusion

Smart cities use digital technologies to enhance performance, well-being and improve the quality of living. This will inevitably require the active participation of citizens. Transport, Energy, Healthcare, Water and Waste are key areas of smart cities' concerns. Smart-GH uses raw open-data collected about air quality, as well as noise measurements collected by individuals, and aggregates them to produce actionable information in the transport domain. This, we believe, is in the highest tradition of open-source and open-data where small individual contributions are combined to produce a social good. Citizens usually invest a substantial chunk of time in commuting, and enabling citizens to improve the quality of this time would directly improve their well-being, and their quality of life. To the best of our knowledge, Smart-GH is the first GIS system that adds such benefits to the traditional usage of common GIS applications.

Acknowledgments. This work was partially supported by the EU Project Diversify FP7-ICT-2011-9.

References

1. Cayo, M., Talbot, T.: Positional error in automated geocoding of residential addresses. International Journal of Health Geographics **2**(1) (2003)
2. D'Hondt, E., Stevens, M., Jacobs, A.: Participatory noise mapping works! An evaluation of participatory sensing as an alternative to standard techniques for environmental monitoring. Pervasive and Mobile Computing **9**(5), 681–694 (2013)
3. Gallagher, R.: Operation Auroragold: How the NSA hacks cellphones worldwide (December 2014). https://firstlook.org/theintercept/2014/12/04/nsa-auroragold-hack-cellphones/
4. Goodchild, M.: Citizens as sensors: the world of volunteered geography. GeoJournal **69**(4), 211–221 (2007)
5. Helbig, N., Cresswell, A., Burke, G., Luna-Reyes, L.: The dynamics of opening government data: A white paper. Centre for Technology in Government, State University of New York, Albany (2012). http://www.ctg.albany.edu/publications/reports/opendata/opendata.pdf
6. IBM Institute for Business Value. Smarter cities for smarter growth (2010)
7. Kitchin, R.: Four critiques of open data initiatives: The Programmable City (November 2013)
8. World Health Organization. Urbanization and health. Bulletin of the World Health Organisation **88**(4) (2010)
9. Zheng, P., Ni, L.M.: Spotlight: The rise of the smart phone. IEEE Distributed Systems Online **7**(3) (2006)

Adoption, Use, and Impact

A Qualitative Study on the Adoption of Open Source Software in Information Technology Outsourcing Organizations

Lakshmanan Ramanathan[1(✉)] and Sundaresan Krishnan Iyer[2]

[1] Birla Institute of Technology and Science, Pilani, India
sanlakit@gmail.com
[2] Infosys Limited, Mysore, India
sunkashyap@yahoo.com

Abstract. The purpose of this paper is to identify the influence of Outsourcing on Open source software (OSS) and further investigate the factors that impact the adoption of OSS in global Information Technology (IT) outsourcing organizations serviced by Indian IT services providers. This exploratory research adopted positivism research philosophy and qualitative approach. An in-depth interview was conducted with ten participants across IT outsourcing organizations, IT service providers, and OSS service providers. The results show that IT outsourcing was not found to have an impact on OSS adoption. However, eight factors including management support and OSS support availability was identified to influence OSS adoption. IT services providers can utilize this research model to increase their understanding of why some IT outsourcing organizations choose to adopt OSS, while seemingly similar ones facing similar market conditions do not.

Keywords: Open source software · OSS adoption · IT outsourcing · TOE · Diffusion of innovation · Indian IT

1 Introduction

Over the past two decades, open source software (OSS) has gained significant momentum and has changed the way software is perceived, developed and deployed. It is often seen as a disruptive technology that has changed the rules of the industry. In India, Information Technology (IT) industry is one of the most significant growth contributors. As a proportion of India's Gross Domestic Product, aggregate IT sector revenues have grown from 1.2% in 1998 to 8.1% in 2014 [1]. A Gartner report highlights that IT outsourcing organizations are compelled to look at OSS alternatives as concerns around security, performance and technical support are increasingly addressed and India-based IT services providers must evolve to capitalize on this OSS trend [2]. In this study, we explore the role of outsourcing in OSS adoption and develop a conceptual model for OSS adoption in Global IT Outsourcing Organizations (Clients) serviced by Indian IT services providers (Vendors). The scope of this study included Indian IT services providers that are members of the National Association of Software and Service Companies (NASSCOM), the industry association for the IT-BPM sector in India and their clients.

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 103–113, 2015.
DOI: 10.1007/978-3-319-17837-0_10

2 Related Work

OSS may be defined as a software that is released under the terms of a license that allows the licensee to use, modify, and redistribute, either gratis or for a fee. Over 75% of IT organizations leverage nontrivial elements of OSS in their mission-critical IT portfolios, including cases where they might not be aware of it [3]. Researchers explored various aspects of OSS over the past decade and a number of special research areas have emerged. Feller et al. [4] analyzed 155 OSS research artefacts and concluded that the literature has large gaps, and that commercial organizations are underrepresented. Stol and Babar [5] reviewed 219 OSS publications and concluded that OSS in organizations attracted limited attention. Likewise, Hauge et al. [6] have done a systematic literature review and concluded that the overall rigor of the studies performed on OSS, both within organizations and in general, is furthermore not good enough. Ven and Verelst [7] investigated the OSS adoption in Belgian organizations based on TOE framework and identified five critical factors (i.e., software cost advantage, switching costs, reliability, presence of boundary spanners, and availability of external support).

2.1 Research Gap

Previous studies conclude that there is a paucity of information in the models, theories, and frameworks to explain the adoption of OSS in organizations. Studies in the past have focused primarily on the OSS development model and the unique aspect of OSS. Having reviewed the previous studies in literature, it is apparent that some major gaps exist in the OSS research with respect to adoption in corporate sector. Studies lack a robust framework that helps organizations for adopting OSS. There are very limited studies on OSS usage in the context of outsourced software engineering process. In addition, there has been little study on OSS adoption in developing countries [8], like India. Even though there has been an increasing commercialization of OSS, only less information is available on adoption of OSS in IT outsourcing and IT services organizations. Raina and Wurster [2] state that Indian IT providers must find ways to coexist with open source by developing an open source revenue model that complements their current offerings in order to increase their market share in OSS space.

2.2 Theoretical Framework

OSS adoption is a form of technology adoption that refers to a process in which the organization associates itself with OSS in one or many forms. This study uses the adoption model proposed by Hauge *et al.* [6] that includes a) using OSS development practices, b) participating in existing OSS development, c) providing OSS products, d) using OSS tools, or e) deploying OSS products. Much of the technology diffusion literature focuses on the adoption decisions of individuals [9]. Hammouda [8] proposed an empirical model for analysing OSS adoption in Tunisian Software Business leveraging Strauss and Corbin's [10] paradigm. However, this study develops the Diffusion of Innovation theory, which is at organization level, and especially the Technology-Organization-Environment (TOE) framework developed by Depietro *et al.* [11]. Since

the aim of this study is to conduct a comprehensive investigation into the factors influencing the adoption of OSS, the TOE framework allows us to consider the broader context in which this adoption process takes place. The importance of taking into account organizational and environmental characteristics has been stressed by several other studies e.g., [12, 13].

Factors Hypothesized to Influence OSS Adoption

***Reliability*:** Studies [14] indicated that increase in reliability of the OSS would enhance the adoption rate among users. The study by Dedrick and West [15] claims that even in larger organizations, reliability played a significant role. This leads to Hypothesis 1 "*IT Outsourcing Organizations that perceive OSS to be reliable will exhibit a larger extent of OSS adoption.*"

***License and Legal concerns*:** Organizations are concerned about the complications that emerge when various OSS components, governed by different licenses, are used in the same software system [16]. Previous studies [17, 2] confirmed this line of thought. This leads to Hypothesis 2 "*IT Outsourcing Organizations that perceive less concern related to OSS licensing and legal issues will exhibit a larger extent of OSS adoption.*"

S***oftware cost*:** Literature states that the less expensive the technology, the more likely it is that it will be adopted [13]. Previous studies [15, 18, 7] perceived OSS as less expensive and influence adoption. This leads to Hypothesis 3 "*IT Outsourcing Organizations that perceive OSS to be less expensive will exhibit a larger extent of OSS adoption.*"

***Management Support*:** OSS should be part of a strategy where management are involved in the decision making process [19]. Several studies confirmed the importance of management support in the adoption of the innovation [20, 19]. This leads to Hypothesis 4 "*IT Outsourcing Organizations in which management support is high will exhibit a larger extent of OSS adoption.*"

Outsourcing was mainly motivated by cost savings, but has now developed into a routine strategic management [21]. This leads to Hypothesis 5 *"IT Outsourcing Organizations in which IT outsourcing is high will exhibit a larger extent of OSS adoption.*"

A*vailability of OSS Support:* Lack of support is identified as an important barrier for OSS adoption. Li *et al.* [22] state that the availability of support did have an influence on OSS adoption. This leads to Hypothesis 6 *"IT Outsourcing Organizations that perceive support for OSS to be available will exhibit a larger extent of OSS adoption."*

Software Vendor Relationship: Structured vendor support should be in place to complement the existing IT support structures [15]. Organizations have created dependency on their vendors which influence OSS adoption [7]. This leads to Hypothesis 7 "*IT Outsourcing Organizations that have a relationship with an OSS vendor will exhibit a larger extent of OSS adoption."*

OSS Support Availability vs Software Cost: Li *et al.* [23] states that the availability of the external human capital for OSS support will reduce switching cost. Evaluation of service providers requires time, effort and financial resources. This leads to Hypothesis 8 "*IT Outsourcing Organizations that perceive support for OSS to be available will perceive the software costs involved in adopting OSS to be lower."*

3 Research Method

The objective of this research is to study the OSS adoption in IT Outsourcing organizations serviced by Indian IT services providers. The study attempts to answer the following research question: What are the enablers/inhibitors of OSS adoption in IT outsourcing organizations serviced by Indian IT service providers? Thus, IT Outsourcing organizations (Clients) are the unit of analysis. The study used an exploratory qualitative and the multiple case study approach (including vendors and clients), which provides a rich and in-depth analysis of OSS adoption decisions of Organization. Table 1 below summarizes the research findings based on the cases sampled using Theoretical sampling strategy [24]. The variations in type, size of organization, position of respondents allowed exploring diverse organizational and environmental issues.

Table 1. Overview of the Organizations in the Qualitative Study

Case	Designation of Interviewee	Company Profile	Company Location	Size^	Type	OSS Adoption level
C1	Senior Vice President & Chief Technology Officer	Leading financial processing services provider in Canada	N.America	Medium	IT Outsourcing Organization	Extensive
C2	Vice President & Head of IT	Private Life Insurance company	India	Large	IT Outsourcing Organization	Nil
C3	General Manager - IT Services	One of Top 15 Indian IT service providers	India	Large	IT Service Provider	Extensive
C4	Associate Vice President & Senior Delivery Manager	One of Top 5 Indian IT service providers	India	Very Large	IT Service Provider	Sporadic
C5	Chief Executive Officer	Open Source software Solutions Service Provider	Europe	Very Small	OSS Service Provider	Extensive
C6	Vice President, Sr. Technology Manager	Multinational banking and financial services corporation	N.America	Very Large	IT Outsourcing Organization	Sporadic
C7	Senior Manager, Portfolio Leader	Multinational IT, consulting service provider	India	Very Large	IT Service Provider	Sporadic
C8	Principal Architect	Leading Insurance major in USA	N.America	Large	IT Outsourcing Organization	Sporadic
C9	Delivery Manager	One of Top 5 Indian IT service providers	India	Very Large	IT Service Provider	Sporadic
C10	Commercial and IP Licensing Lawyer	Consulting company in OSS/embedded systems	India	Very Small	OSS Service Provider	Sporadic

^ No. of Employees Very Small (<100), Small (101-1000), Medium (1001-10,000), Large (10,001-100,000), Very Large (>100,000)

This study ensured construct validity by reconciling multiple sources of evidence (triangulation) such as multiple case study and OSS literature and reports related to OSS [24, 25]. Further the case study process (semi-structured interviews) by selecting

the concepts to be studied for this research from the literature. Additionally self-selection bias was eliminated by ignoring responses from: a) participants who are not from the specified NASSCOM member list, and b) participants whose company names are not available/who did not reveal company names.

3.1 Within-Case Analysis

The initial list of codes was defined based on the factors that were identified during the literature review. The transcripts were coded to determine factors that influenced the OSS adoption decision. Data pattern-matching [25] was used to identify relevant text extracts for each factor. The data analysis process was flexible and opportunistic [24, 25], wherein new adoption factors were identified iteratively. Data displays were used to summarize and analyze the qualitative data. For each case, a table was constructed that provided an overview of the perception of the organization toward the various adoption factors.

3.2 Cross-Case Analysis

Cross-case analysis allows to compare factors across all cases and then to select the most logically replicated and generalizable factors [24]. 22 codes emanated from the eight factors identified in the literature review in addition to eight new codes that emanated from the cases. Table 2 shows all the factors identified in with-in case analysis. Factors with a minimum frequency count of four cases were identified to determine which factors had an important influence on the organizational adoption decision.

Table 2. Frequency Analysis of Factors

Context	Factors	Cases	Frequency (* >= 4)
Technological Context	Reliability	C1↑, C2↑, C3↑, C5↑, C6↑, C7↑, C8↑, C9↑	**8***
	License Concern	C4↓, C8↓, C9↓, C10↓	**4***
	Legal Concern	C1↓, C2↓, C8↓, C9↓, C10↓	**5***
	Software Cost	C1↓, C2↓, C3↓, C5↓, C6↓, C7↓, C8→, C9↓, C10↓	**9***
Organizational Context	Management Support	C1↑, C2↑, C3↑, C4↑, C6↑, C7↑, C8↑, C9↑	**8***
	IT Outsourcing	C1↑, C10→	2
Environmental Context	OSS Support Availability	C1↑, C2↑, C3↑, C6↑, C7↑, C8↑, C9↑, C10↑	**8***
	Software Vendor Relationship	C1↑, C2↑, C6↑, C8↑	**4***

↑ Enabler to OSS Adoption ↓ Inhibitor to OSS Adoption → No impact/neutral on OSS Adoption

4 Discussion of the Findings

Fig 1. below depicts the summary of case study findings.

4.1 Enablers of OSS Adoption

Reliability: Eight organizations indicated that the high reliability of OSS was an important factor in the adoption decision. Three organizations (C1, C5, C7) stated that OSS was highly reliable and they used OSS in production, whereas five organizations (C2, C3, C6, C8, C9) highlighted lack of reliability for not adopting OSS. Reliability was expressed in terms of the following items: security, stability/scalability/ maturity, and lack of features. The participant in case C5 mentioned that: *"system is afterwards (of OSS deployment) [...] more efficient to manage, more stable. So, it is really worth it"*. The finding of the study is in line with the previous studies which indicated that an increase in reliability of OSS would enhance the adoption rate among users [14]. Consequently, the technology context attribute *Relative Advantage* was seen in terms of *Reliability*, and played an important role in OSS adoption.

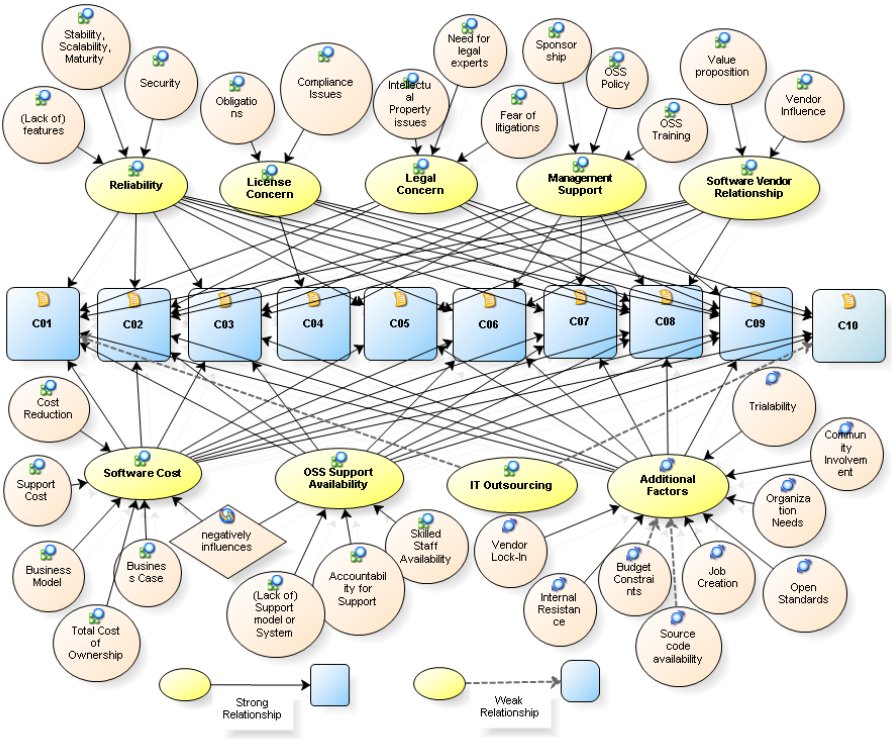

Fig. 1. OSS Factors in Case Study Findings

Software Cost: Nine organizations indicated that the software cost was an important factor in the adoption decision. Software cost was expressed in terms of the following items: business case, business model, support cost, total cost of ownership, and Cost reduction. While some of the wealthiest banks moved towards OSS to save costs (C5), couple of organizations (C7, C9) considered cost as a trade-off and not as a decision making parameter. Cost reduction and lower total ownership cost of the software including exit costs of proprietary software and ongoing support costs was mentioned as a critical factor in OSS adoption. Several studies [15, 18, 7] perceived OSS as less expensive and hence have an influence on OSS adoption. Consequently, the technology context attribute *Relative Advantage* was seen in terms of *Software cost*, and played an important role in OSS adoption.

Management Support: Eight organizations indicated that management support was an important factor in the adoption decision. Management support was expressed in terms of the following items: OSS training, OSS policy, and sponsorship. OSS training could increase the knowledge quotient of employees and subsequently increase adoption (C4, C7). Six organizations (C2, C3, C4, C6, C7, C9) discussed about role of OSS policy while organizations (C1, C2, C3, C6, C9) discussed about role of sponsorship in OSS adoption. Management team acted as a *Boundary spanner* to evangelize OSS initiative across the organization. Consequently, the organization context attribute *Boundary spanners* was seen in terms of *Management support*, and played an important role in OSS adoption.

Availability of Support: Eight organizations indicated that OSS support availability was an important factor in the adoption decision. OSS support availability was expressed in terms of: accountability for support, lack of support model or system, and skilled staff availability. Three organizations (C1, C2, C7) discussed about accountability for support of OSS. The participant in case C1 mentioned that: "*The auditor wants to know if a system breaks how you are going to get support*". Four organizations (C1, C3, C8, C10) emphasized lack of availability of skilled staff for OSS. Consequently, the environment context attribute *Support infrastructure* was seen in terms of *OSS support availability*, and played an important role in OSS adoption.

Software Vendors: Four organizations indicated that the software vendor influences OSS adoption. Software vendor relationship was expressed in terms of: vendor influence, and value proposition. Organizations were concerned about smaller vendors offering OSS support, wherein the net worth of the vendor was many times less than the indemnification value and hence were perceived to be unstable (C2). This was highlighted by Ven & Verelst [14], wherein the long-term viability of small organizations were questioned. Consequently, the environment context attribute *Network effects* was seen in terms of *Software vendor relationship* and played an important role in OSS adoption. Further, case study provided additional insights into the OSS adoption by identifying other factors that affect OSS adoption (such as OSS community involvement, organization needs, open standards, and job creation).

4.2 Inhibitors of OSS Adoption

Four organizations indicated that the license concerns impacts OSS adoption. License concern was expressed in terms of: compliance issues, and obligations. The participant in case C10 mentioned that: "*… must be compliant with license attached to that (OSS) software […] need to check permissive license or restrictive license*". All four organizations highlighted concerns related to OSS license obligations. Five organizations indicated that the legal concerns hinder OSS adoption. Legal concerns were expressed in terms of: intellectual property issues, fear of litigations, and need for legal experts. Two organization (C1, C2) highlighted fear of litigations. For instance, the participant in case C1 mentioned that: "*The one thing you can't stop (in OSS) is (…) litigation*". Case C8, C9, C10 discussed about the needs for legal experts. This suggests that clients had limited expertise in understanding the OSS legal nuances and needed legal experts. The need for legal expertise was also highlighted by Hammouda *et al.* [16] that stated some of the OSS licenses were fundamentally incompatible with each other. Consequently, the technology context attribute *Compatibility* was seen in terms of *License and legal concerns*, and played a role in inhibiting OSS adoption. **Organization size** was chosen as a moderating variable in this study. However, the findings from qualitative analysis do not support a relationship between size of organization and OSS adoption. While this contradicts the study by Fichman [12] that stated organization size has frequently been found to have a positive impact on the assimilation of new technologies, a possible explanation could be that smaller organizations have fewer resources and might adopt OSS to reduce costs. Further, case study provided additional insights into the OSS adoption by identifying other factors that hinder OSS adoption (vendor lock-in, internal resistance, and lack of support).

4.3 Factors That Do Not Impact OSS Adoption

IT outsourcing: Only two organizations mentioned about the role of IT outsourcing in OSS adoption. IT service providers were just soliciting advice on OSS. Given that only two cases reported IT outsourcing, the organization context attribute *Formalization* in terms of *IT Outsourcing* was not found to have an impact on OSS adoption. Further, case study provided additional insights into the OSS adoption by identifying other factors that do not impact OSS adoption (like budget constraints, source code availability, and trialability).

5 Conclusion

The present research contributes to the organizational adoption literature by exploring the adoption of OSS in IT outsourcing organizations serviced by Indian IT service providers. To investigate this research problem, the study proposed a conceptual

model that describes a number of factors which were hypothesized to influence the adoption of OSS. The perceptions of different organizations differed based on their needs and their clients' requirements. The findings summarizes that the OSS products which were highly reliable and mature were used in production servers in a significant way. In addition, OSS product must have desired minimum features and a roadmap for continuous improvement compared to similar proprietary software. Cost savings was an important factor in enabling OSS adoption. The perceived litigations/IP issues were hindrance to OSS adoption. Management had to deal with many issues in the OSS adoption decision process including career path for internal support team, indemnification issues, capital expenses vs. operational expenses, higher cost for external support etc.,. Lack of defined OSS support model and non-availability of skilled staff, was a hindrance for OSS adoption. Technology innovation requires organizations to simultaneously 'change' to fix and improve the past as well as 'transform' to create a futuristic vision. While factors like Reliability, Software Cost, Management support etc. can be classified as 'change' category, factors like Software vendor relationship, Organizational needs, Availability of support, would be classified as 'transformation' category.

5.1 Theoretical Contributions and Implications

There are limited studies on OSS adoption in the context of outsourcing software development process and many studies were focused on the management aspect of developing software. In addition, empirical findings obtained from the present study will contribute to the literature on OSS adoption in Indian outsourcing organization, an area where empirical studies are scant. The framework can be used by Indian IT services providers to better frame their strategies to service their clients. IT services providers can use this research model to increase their understanding of why some IT outsourcing organizations choose to adopt OSS, while seemingly similar ones facing similar market conditions do not. IT services providers can offer "OSS as a service" for its clients and help them address the gaps in support availability and achieve reduction in total cost of ownership of software.

5.2 Limitations and Future Research

The main limitation of our research is that it is focused on IT outsourcing organizations serviced by Indian IT services providers. Hence, we cannot safely generalize our finding to other regions. We decided to use the TOE framework as theoretical base. However, the use of a stronger theoretical framework could have provided a richer insight in our data. Therefore, it would be interesting, if future studies try to build on the results from this study and study the adoption of OSS using a strong theoretical foundation. Since this study encompasses OSS in general, future studies could also determine if our results are also applicable to all types of OSS.

Appendix

Themes used in semi-structured interview questions

- How and to what extent is the (client) organization currently facilitating in adopting OSS?
- OSS usage within the organization (Success stories/Failures in OSS Adoption)
- Experience about the availability of support and maintenance of OSS products
- Impact of factors identified in literature on OSS adoption in the organization
- IT Service providers' role in OSS adoption strategy (For IT Outsourcing organizations)
- IT Service providers' strategy with respect to OSS (For IT service provider)

References

1. NASSCOM Research. The IT-BPM sector in India - Strategic Review 2014. NASSCOM (2014)
2. Raina, A., Wurster, L.F.: Open source software adoption becoming mainstream in India. Gartner (2013)
3. Driver, M.: Drivers and incentives for the wide adoption of open source software. Gartner (2012)
4. Feller, J., Finnegan, P., Kelly, D., MacNamara, M.: Developing open source software: a community-based analysis of research. In: Trauth, E., Howcroft, D., Butler, T., Fitzgerald, B., DeGross, J. (eds.) Social Inclusion: Societal and Organizational Implications for Information Systems. IFIP, vol. 208, pp. 261–278. Springer, Boston (2006)
5. Stol, K.-J., Babar, M.A.: Reporting empirical research in open source software: the state of practice. In: Boldyreff, C., Crowston, K., Lundell, B., Wasserman, A.I. (eds.) OSS 2009. IFIP AICT, vol. 299, pp. 156–169. Springer, Heidelberg (2009)
6. Hauge, Ø., Ayala, C., Conradi, R.: Adoption of open source software in software-intensive organizations - A Systematic literature review. Information and Software Technology **52**(11), 1133–1154 (2010)
7. Ven, K., Verelst, J.: A Qualitative study on the organizational adoption of open source server software. Information Systems Management **29**(3), 170–187 (2012)
8. Hammouda, I.: Open source software in tunisian software business: an empirical study. In: EUROMICRO-SEAA, pp. 451–454 (2010)
9. Oliveira, T., Martins, M.F.: Literature review of information technology adoption models at firm level. The Electronic Journal Information Systems Evaluation **14**(1), 110–121 (2011)
10. Corbin, J., Strauss, A.: Basics of qualitative research: Techniques and procedures for developing grounded theory. SAGE Publications (2014)
11. Depietro, R., Wiarda, E., Fleischer, M.: The context for change: organization, technology and environment. In: Tornatzky, L.G., Fleischer, M. (eds.) The Processes of Technological Innovation, 1st (edn.), pp. 151–175. Lexington Books, Massachusetts (1990)

12. Fichman, R.G.: The diffusion and assimilation of information technology innovations. In: Markus, M.L., Tanis, C., Zmud, R.W. (eds.) Framing the Domains of IT Management: Projecting the Future Through the Past, pp. 105–127. Pinnaflex Educational Resources, Ohio (2000)
13. Rogers, E.M.: Diffusion of Innovations, 4th (edn.), pp. 219–287. Simon and Schuster, New York (2010)
14. Ven, K., Verelst, J.: An empirical investigation into the assimilation of open source server software. Communications of the ACM **28**(1), 9 (2011)
15. Dedrick, J., West, J.: Why firms adopt open source platforms: a grounded theory of innovation and standards adoption. In: Proceedings of the Workshop on Standard Making: A Critical Research Frontier for Information Systems, MIS Quarterly Special Issue Workshop, Seattle, WA, pp. 236–257 (2003)
16. Hammouda, I., Mikkonen, T., Oksanen, V., Jaaksi, A.: Open source legality patterns: architectural design decisions motivated by legal concerns. In: Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments, New York, NY, USA, pp. 207–214. ACM (2010)
17. Fitzgerald, B.F., Bassett, G.: Legal Issues Relating to Free and Open Source Software, vol. 1, pp. 11–36. Queensland University of Technology, Brisbane (2004)
18. Spinellis, D., Giannikas, V.: Organizational adoption of open source software. Journal of Systems and Software **85**(3), 666–682 (2012)
19. Hauge, Ø., Cruzes, D.S., Conradi, R., Velle, K.S., Skarpenes, T.A.: Risks and risk mitigation in open source software adoption: bridging the gap between literature and practice. In: Ågerfalk, P., Boldyreff, C., González-Barahona, J.M., Madey, G.R., Noll, J. (eds.) OSS 2010. IFIP AICT, vol. 319, pp. 105–118. Springer, Heidelberg (2010)
20. Glynn, E., Fitzgerald, B., Exton, C.: Commercial adoption of open source software: an empirical study. In: Proceedings of International Conference on Empirical Software Engineering, Noosa Heads, Australia, pp. 225–234. IEEE (2005)
21. Hoecht, A., Trott, P.: Innovation risks of strategic outsourcing. Technovation **26**(5), 672–681 (2006)
22. Li, Y., Tan, C.-H., Xu, H., Teo, H.-H.: Open source software adoption: Motivations of adopters and amotivations of non-adopters. ACM SIGMIS Database **42**(2), 76–94 (2011)
23. Li, Y., Tan, C.-H., Teo, H.-H., Siow, A.: A human capital perspective of organizational intention to adopt open source software. In: Proceeding of the 26th Annual International Conference on Information Systems (ICIS 2005), Las Vegas, NV, USA, pp. 137–149 (2005)
24. Yin, R.K.: Case study research: Design and methods, 5th (edn.), pp. 67–162. SAGE Publications, California (2013)
25. Eisenhardt, K.M.: Building theories from case study research. Academy of Management Review **14**(4), 532–550 (1989)

Surveying the Adoption of FLOSS by Public Administration Local Organizations

Davide Tosi[1(✉)], Luigi Lavazza[1], Sandro Morasca[1], and Marco Chiappa[2]

[1] Università degli Studi dell'Insubria, DISTA, Via Mazzini, 5, Varese, Italy
{davide.tosi,luigi.lavazza,sandro.morasca}@uninsubria.it
[2] Consiglio Regionale della Lombardia, Via F. Filzi, 22, 20124 Milano, Italy
marco.chiappa@gmail.com

Abstract. Background. The introduction of Open Source Software technologies in the Public Administration plays a key role in the spread of Open Source Software. The state of the art in the adoption of Open Source Software solutions in the Public Administration is not very well known even in areas like Lombardy, which is Italy's largest and most developed region.

Goal. The goal of the investigation documented in this paper is to obtain a clear picture about the introduction of Open Source Software technologies in the Public Administration, the obstacles to their adoption, and the willingness of stakeholders to proceed with their introduction.

Method. We carried out a qualitative and quantitative survey that was submitted to a representative part of the Public Administrations in Lombardy.

Results. The analysis of the qualitative and quantitative information shows that several Public Administrations are already using Open Source Software technologies, though not in all application areas. The savings are one frequently cited incentive to the adoption of Open Source Software. However, one obstacle is the fact that a comprehensive law on software in the Public Administration has not yet been approved.

Conclusions. Our analysis provides results that indicate a common understanding of incentives, obstacles, and opportunities for Open Source Software technologies in Public Administrations.

Keywords: Public administrations · FLOSS adoption · Survey · Italy

1 Introduction

Transparency, *reuse*, and *participation* are the final goals of Open-Government, based on the four technological cornerstones [1]: *Open Source Software*, *Open Format* and *Open Data* [2] in a context of infrastructure and hosting that is as Open as the *Open Cloud* [7].

In the last few years, there is a slowly increasing interest by Italian public and private entities in the world of Free-Libre Open Source Software (FLOSS) [4]. On the one hand, the increasing level of FLOSS product quality is also increasing the trust that end users have in FLOSS products. On the other hand, the need for budget cuts to contain costs and expenditures of public and private entities provides new motivations for the adoption FLOSS at the expense of commercial proprietary software products.

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 114–123, 2015.
DOI: 10.1007/978-3-319-17837-0_11

The global importance of FLOSS worldwide has also led to the proposal and approval of a number of laws whose goal is to regulate and possibly favour the use of FLOSS in the Public Administration Local Organizations (PALO). For instance, at the Italian national level, Decree No. 267/2000 [5] and directive 19/12/2003 [6] have been adopted to regulate the use of software in PALO.

The Region of Lombardy, with Law Proposals on "Rules on information technology pluralism and adoption of open formats and standards for digital documents in the information society of Lombardy" and on "Provisions on access, publishing and reuse of public data and the regional administration in open format using free software and the Internet" has given a strong signal of openness to FLOSS and Open Data [2, 3]. To obtain concrete outcomes for the effort spent by the Region of Lombardy and the political Regional Council groups, it is necessary to continuously promote the activities related to FLOSS and Open Data with large projects, for instance with the definition of an Observatory on best practices in the use of FLOSS and the publication of Open Data.

This paper describes the execution of a qualitative and quantitative survey for assessing the state of the art in the introduction of FLOSS in the PALO in Lombardy. Specifically, the survey addressed a fairly large sample of the PALO of the Region of Lombardy, as it included municipalities that account for about one-fifth of the population of the region. The survey investigates the degree of use of FLOSS technologies in the region, the obstacles to its introduction, and the willingness of stakeholders to introduce FLOSS. The analysis of the quantitative data from the survey confirmed the qualitative data.

Lombardy is by far Italy's largest region in terms of population and arguably the most economically and technologically advanced region of the country too. So, the large-scale introduction of FLOSS technologies in Lombardy would amount to a large-scale introduction of FLOSS technologies in the country and would propel the introduction of FLOSS in the other parts of Italy.

Several surveys have been carried out worldwide to monitor and understand the diffusion of FLOSS. A fairly comprehensive systematic literary survey concerning the adoption of FLOSS in software-intensive organizations (not necessarily PA) can be found in [11]. The adoption of FLOSS in Venezuela was studied and reported in [12]. The factors that affect the adoption of FLOSS in public organizations was studied in [13]. Although not addressing PA, the framework provided in [13] can be used to explain several of our findings, thus showing that the concerns that affect the adoption of FLOSS in PA are not dissimilar from those affecting industrial contexts.

The remainder of this paper is organized as follows. Section 2 describes the methodology adopted to conduct the quantitative and qualitative survey. Sections 3 and 4 report on the results of the qualitative and quantitative results, respectively. We discuss the results and conclusions in Section 5.

2 Methodology

The survey about the adoption of FLOSS by PALO in Lombardy was based on a) the qualitative analysis of a sample of medium-large organizations and b) the quantitative analysis of all organizations located in Lombardy. Thus, we obtained both qualitative

semantically valuable –though not immediately generalizable– information as well as objective and statistically representative data, though possibly less detailed and contextualized.

Qualitative analysis was performed in PALO already acquainted with FLOSS. A questionnaire containing semi-structured open questions was used as a basis for collecting opinions and indications concerning FLOSS from people in charge of IT development and operation. The interviews carried out by means of the qualitative questionnaire aimed at understanding and describing the procedures and best practices used to launch development or migration projects based on FLOSS and at determining the factors that affect (either positively or negatively) the adoption of FLOSS. Accordingly, the interviewer concentrated on understanding the critical factors that characterize the adoption of FLOSS, how risks are managed, and what guidelines are followed in the adoption of FLOSS. In any case, the interviewer also explored additional issues that would be raised during the interviews. The interviews were performed in the municipalities of Bollate, Brescia, Cinisello Balsamo, Milano, Monza, Vigevano, and the Province of Lecco, which range from approximately 36,500 to approximately 1,350,000 inhabitants.

The quantitative analysis was carried out via a questionnaire containing 35 closed-answer questions, defined on the basis of the results obtained by the qualitative analysis. The objective was to exhaustively check the situation of PALO in Lombardy, so, the questionnaire was sent to all the municipalities in Lombardy. The questionnaire was implemented via the FLOSS platform LimeSurvey [www.limeservice.com], which greatly helped users in the filling out the questionnaire and researchers in data collection and analysis. The questionnaire addressed specific issues concerning FLOSS, e.g., the pros and cons of adopted FLOSS solutions, the FLOSS tools being used, the cost of management and the more frequently used open source licenses. The invitation to answer the questionnaire was sent to all the 1536 municipalities in Lombardy on September 9, 2012. Two reminders were sent on October 17 and November 20. The questionnaire was closed on December 31, 2012. 451 questionnaires were returned, of which 256 compiled completely. The received answers were checked to eliminate typos and inconsistencies due to possible misinterpretations.

The received answers account for municipalities with a total of 1,927,189 inhabitants (about 19% of total Lombardy inhabitants), therefore they are a statistically relevant sample. Also the population distribution in the respondent municipalities matches the population distribution in Lombardy municipalities, which is characterized by many municipalities with medium-sized population. While data of this survey refer to 2012, we believe it is still a valid picture of the current situation of Lombardy. Due to the election of a new regional President and Council (mid 2013), the adoption process of FLOSS was slowed down considerably.

3 Results of the Qualitative Analysis

Seven professionals in charge of IT development and operations were interviewed. They provided a quite clear and complete view of FLOSS and the pros and cons of adopting FLOSS in a PALO. They reported a clear propensity to use FLOSS.

However, they also highlighted the importance of carefully and completely analysing the requirements of the problem at hand to achieve the best solution, be it open source or proprietary.

The adoption of FLOSS in the Italian PA began quite recently, and the process of migrating to FLOSS is still largely unexplored. This situation is probably caused by a very little knowledge of FLOSS and the implications of adopting FLOSS. Moreover, most software platforms in use by the PALO are proprietary monolithic applications that appear difficult to replace or even to integrate with FLOSS. In particular, it appears difficult to integrate FLOSS-based CRM (Customer Relationship Management), DMS (Document Management System), ERP (Enterprise Resource Planning) or network and service monitoring. On the contrary, it seems that the adoption from scratch of completely new platforms and applications can be pursued both via open and proprietary solutions. In the latter cases, the decision to what extent OSS should be adopted is driven by several factors:

- Personal curiosity of IT people with respect to open solutions that could break the "de facto standard" created by multinational companies like Microsoft;
- The political orientation of the administrations;
- Technical considerations concerning qualities like usability, reliability, etc.

Interviewees also stressed that the process of adopting FLOSS is very different for server-side and client-side software. Using FLOSS on the servers is easier –being transparent to both PA employees and citizens– even though installing, configuring and managing OSS software (like Linux, or an open source email server, or an open source Content Management System supporting the municipal Web portal) requires a bigger effort than the proprietary alternatives.

When client software is concerned, the situation is very different. Although mature applications that could be used instead of the proprietary counterparts –like Open office or Gimp– are available, PA employees are not willing to change, and training people to use the new software is a long and expensive process. Moreover, since proprietary file formats (like .doc document files) are widely used by both the PA and the citizens, a seamless and correct conversion of a huge amount of documents is also necessary but difficult to achieve. Finally, the lack of resources makes it difficult to carry out education and dissemination initiatives to promote the usage of FLOSS, especially in schools and in public organizations where cultural "digital divide" is greatest [10].

The advantages of FLOSS for the PA and the factors that are perceived to limit the adoption of FLOSS, according to the interviewees, are listed in Table 1 (in order of decreasing importance).

To overcome the problems listed above, several interviewees advocated region-wide guidelines and rules that drive the adoption of FLOSS in PALO. Such rules and guidelines should also clarify national laws and dispositions [5,6]. Funding of projects addressing the adoption of FLOSS is also deemed necessary.

Table 1. Reported Pros and Cons in adopting FLOSS

	Pros	Cons.
1	Money savings	Need for training
2	Security and stability of applications	Resistance to change
3	Open and standard data formats	Difficulty to find professionals that can support the adoption and/or migration process
4	Continuous technological update	Difficult integration with proprietary software
5	Support by FLOSS development communities	No funding from the regional administration
6	Possibility of customizing and reengineering FLOSS	Very specific PALO needs are not addressed by FLOSS
7	-	Difficult conversion to/from proprietary formats

4 Results of the Quantitative Analysis

The IT and Data Centre departments of the interviewed PALO are medium-sized departments, with an average of 46 client machines (with a maximum of 1900 client machines) and an average of 3 server machines at the infrastructural level (with a maximum of 70 server machines). The Milan municipality is the one with the highest weight in this survey. Out of 218 municipalities that responded to question "Do you already Adopt Free and Open Source Software (FLOSS) in your Organization?" 38% does not use any type of FLOSS, while 50% (108 respondents) use FLOSS. The remaining (13%) does not know or gives no response. Out of 108 PALO that adopt FLOSS solutions, the majority declares to use FLOSS software only partially on desktop computers and servers. Very few PALO use FLOSS on desktop computers and servers more than proprietary software solutions. Nine PALO declare to have tested Software FLOSS only on pilot projects.

As clearly shown in the chart of Figure 1, the most common FLOSS solutions are in the software category: "Browser" (20.13%), with a strong preference for the Mozilla Firefox browser, or in the category: "Productivity Software" (17.90%) with a preference for the OpenOffice package. Very adopted are also: Data Base Management Systems (11.63%), with an equal distribution between MySQL and PostgreSQL, Operating System (10.51%) such as Linux, Mail Client (10.29%) such as Zimbra or Mozilla Thunderbird. The qualitative survey shows clearly that solutions for Network and IT Services Monitoring, CRM systems (Customer Relationship Management), DMS (Document Management System), ERP (Enterprise Resource Planning) and Networking Systems are slightly adopted in PALO. This situation is in contrast with the policy of transparency and dematerialization that PALO should or would like to pursue. As for Operating Systems on PC Clients and Servers, there is a clear predominance of proprietary solutions (i.e., Microsoft Windows): 198 out of 210 operating system installations are based on Microsoft solutions from a client point-of-view (4 Linux distributions and 2 MacOS systems); as for server machines, 191 out of 242 are Microsoft installations (41 Linux distributions and 0 MacOS systems). However, proprietary solutions adopted on Server are out of date (i.e., Windows Server 2003): this means that the monolithic infrastructure designed several years ago are not constantly updated and maintained. Proprietary solutions adopted on PC Clients (such as

Windows XP) are often obsolete as well, since the regular upgrade of the proprietary OS is too expensive for the finances of the municipal administrations.

The considerations listed above are confirmed by the desire of mayors and IT municipal managers to increase the use of FLOSS software in their organizations. 54% of respondents would like to adopt new FLOSS solutions, while only 9% are not interested in adopting FLOSS (31% do not know, and 6% do not answer.)

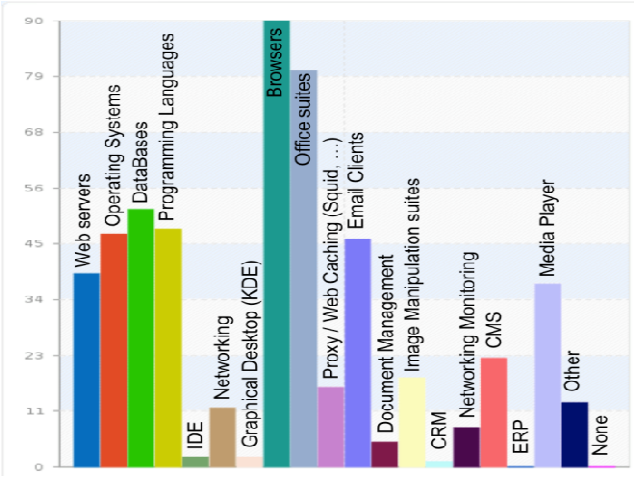

Fig. 1. Micro categories of adopted FLOSS Solutions

A significant proportion of respondents (18%) would like to migrate all of their proprietary software to FLOSS systems, while 74% are interested in migrating only some specific FLOSS solutions (3% do not know, and 5% do not answer.)

However, the migration processes and the ability to access the source code of the adopted solutions are not considered by IT managers as a way to improve their IT department significantly. Only 20% of respondents, in fact, think that access to the source code of the software used by the municipal organizations may be a mechanism to improve the quality of their IT department, while 32% do not see any relation between the access to the source code and the quality of the IT department (40% do not know, and 8% do not answer.)

To identify the strengths and weaknesses in the adoption of FLOSS solutions for the Lombardy PALO in particular, the interviewed PALO were asked to express specific opinions about some qualitative factors of FLOSS solutions: Table 2 summarizes the results for each quality factor expressed both as percentage and absolute values. Among the different quality factors, the aptitude to customize FLOSS solutions compared with proprietary solutions is considered strategic. Out of 218 respondents, 88% completely agree (or agree) that FLOSS is easier to customize than proprietary Software. Another strategic factor for FLOSS is the ease of integration of the FLOSS Software with proprietary software (44.40% completely agree or agree). The openness of the source code is not enough to adopt FLOSS products: lower prices than the equivalent proprietary solutions are sought (41.67% completely agree or agree).

For 36.57% of the respondents, it is difficult to find specialized companies to support the migration to FLOSS and for 41.2% of the respondents, it would be too expensive and too long to train their employees to use FLOSS software. In addition, the respondents think that the proprietary software is easier to use, safer and more reliable than FLOSS [9].

Table 2. Opinions about Qualitative Factors of Software FLOSS

	Fully agree	Agree	Disagree	Fully disagree	No opinion / No answer
a) FLOSS is easier to use than Proprietary Sw	0.9% (2)	22.5% (49)	28.0% (61)	2.3% (5)	37.2% (81) 9.2% (20)
b) FLOSS is easier to customize than Proprietary Sw	5.5% (12)	34.9% (76)	14.7% (32)	0.9% (2)	34.9% (76) 9.2% (20)
c) FLOSS is more reliable than Proprietary Sw	4.1% (9)	19.7% (43)	28.0% (61)	0.9% (2)	38.1% (83) 9.2% (20)
d) FLOSS is more secure than Proprietary Sw	6.0% (13)	19.7% (43)	27.1% (59)	0.9% (2)	36.7% (80) 9.6% (21)
e) The overall quality of FLOSS is higher than the quality of Proprietary Sw	1.4% (3)	16.1% (35)	31.2% (68)	2.3% (5)	39.4% (86) 9.6% (21)
f) FLOSS can be easily integrated with Proprietary Sw	2.3% (5)	32.1% (70)	19.3% (42)	3.7% (8)	33.5% (73) 9.2% (20)
g) If FLOSS would provide only the ability to access the source code without being cheaper than Proprietary Sw, then my PALO would not use FLOSS	11.1% (24)	30.6% (66)	11.6% (25)	1.4% (3)	35.2% (76) 10.2% (22)
h) It is quite hard for my PALO to find companies that provide technical support for FLOSS products	9.3% (20)	27.3% (59)	21.3% (46)	2.8% (6)	29.2% (63) 10.2% (22)
i) I would like to migrate to FLOSS if and only if other organizations already migrated to FLOSS	5.6% (12)	27.3% (59)	29.2% (63)	2.8% (6)	25.5% (55) 9.7% (21)
l) Train the staff of my organization to use FLOSS would be to expensive in terms of cost and time	6.9% (15)	34.3% (74)	24.5% (53)	4.2% (9)	20.4% (44) 9.7% (21)

It is interesting to note that, although a significant percentage of IT managers believe that FLOSS is more secure, (25.68%), more reliable (23.85%) and with a higher overall quality (17.44%) than equivalent proprietary and closed software solutions, only 11% of respondents claim to buy new hardware (PC desktop, laptops, etc.) with the Operating System and the main SW packages already installed and then replace the software over time with equivalent FLOSS solutions. In alternative, they directly buy the hardware without pre-installed software to provide the IT staff with the possibility of installing and configuring FLOSS solutions from scratch. 65% of respondents buy new hardware with pre-installed proprietary Operating Systems and software packages, then add FLOSS solutions over-time whenever new specific needs rise in the PALO. In all cases, the pre-installed software (licensed with the hardware) is not replaced. The Microsoft monopoly in licensing new hardware with their Operating Systems and Office suite increases the hesitation in migrating to FLOSS solutions when the software is purchased with the hardware.

The choices on the purchase of software are more influenced by IT managers (17.78%) that decide which Software solutions purchase. In the decision process, consultants who work with the administration can have an important role (13.78%), such as end-users who, with their needs, indirectly push the administration to make certain choices rather than others (14.44%). With an average of almost 4 suppliers and a total of 664 suppliers, which the Public Administrations rely on to supply Software solutions, the responding PALO said they were too dependent on their suppliers in 23.59% of cases, while they declare their freedom from their suppliers in 47.18% of cases. This indicates how the organizations still feel free to act in their own internal choices regardless of the supplier, while confirming the outsourcing of development and maintenance of the solutions in use. For example, in 54.88% of cases, respondents say they rely regularly and frequently on external suppliers for maintenance of their software, while only 34.36% say they do it sporadically.

In 21% of cases, software solutions are released by vendors as "turnkey" products without requiring ad-hoc customizations and personalization. However, in 46.15% of cases, sporadic customization activities of the solutions adopted are requested by the PALO. Frequent tasks of customization are needed by 7.18% PALO. During the customization of the software solutions, the contribution of the FLOSS community is evaluated by 42.56% of the respondents as “good”, in line with the quality perception PALO IT managers have about the contributions made by developers of proprietary software solutions (51.28%). It is important to notice that 2.05% of the respondents declare an insufficient support provided by FLOSS communities. The value increases to 3.08% in case of the support provided by developers of proprietary software solutions.

In 2011, the total budget spent by the IT departments of the 140 respondents was equal to € 7.7M (million) with an average value of € 55k (thousand) and a peak of € 2M. Analysing the data point with the highest value (i.e., budget > € 100k) and comparing these data with the responses to question "Does the PALO adopt FLOSS?" and the number of inhabitants of the related PALO, we can observe that:

- Only 12 municipalities have an expense of the IT department > € 100k per year;
- The total expenditure of the 12 municipalities is about € 4.5M for their IT departments with € 377k in average, covering 58.45% of the total expenditure;
- The total number of inhabitants of the 12 analysed municipalities is equal to 628,000 inhabitants (the IT expenditure grows linearly with the number of inhabitants), covering 32.60% of the total number of people reported in Table 1;
- 10 out of 12 municipalities under analysis adopt FLOSS solutions, while only 2 municipalities do not use FLOSS (one of these is the municipality with the highest IT departmental expenditure).

In 23.16% of cases, IT managers declare that the total expenditure, in 2011, for their IT department is too high, while 50% of the IT managers claim that the total expenditure was reasonable. Only 1% of the IT managers say that the total expenditure is too low. In any case, 40% of respondents state that they will focus on a reduction of their total IT expenditure in the next two years, while for 26.84% of respondents, a contraction in the IT budget is perceived as not necessary.

59.41% of the respondents to question "Do you feel a need for addressing (directives / laws) at the regional level on the adoption of Open Source Software in PA?" advocates the adoption of a directive or law on the issues of FLOSS. Only 17.65% of the respondents do not see any legislation as necessary. Very similar is the distribution of respondents about a regional directive / law related to Open Data.

5 Conclusions

The results yielded by the qualitative analysis were substantially confirmed by the quantitative analysis performed in the subsequent phase of the investigation. It is therefore possible to conclude that:

- A relevant fraction of the interviewed PALO is already using FLOSS;
- In general, both FLOSS and proprietary software are used. However, the usage of FLOSS is dominant with respect to proprietary software only in 10% of the PALO that participated in survey;
- Sophisticated FLOSS solutions, e.g., for CRM, DMS or ERP, are rarely used in PALO. On the contrary, OS, CMS, productivity suites and Web browsers are more widely used;
- Operating Systems are mainly proprietary. 17% of the interviewed PALO uses Linux on servers, and only 2% on clients;
- Money saving is an important driver for the adoption of FLOSS, which is generally considered of lesser quality than proprietary software by people in charge of IT operations;
- 60% of the interviewees are waiting for a Regional law that prescribes how to adopt FLOSS and how to publish Open Data. To this end, the creation of a regional board for monitoring FLOSS and supporting its adoption is advocated.

Acknowledgements. The research presented in this paper has been partially funded by the FP7 Collaborative Project S-CASE (Grant Agreement No 610717), funded by the European Commission and by project “Metodi, tecniche e strumenti per l’analisi, l’implementazione e la valutazione di sistemi software,” funded by the Università degli Studi dell’Insubria. We are grateful to all IT Managers whom participated to the survey and to Fabio Pizzul and Regional Group of the Democratic Party.

References

1. Open Government Partnership: Action Plan Italiano, April 2011. http://goo.gl/qybte
2. Berners-Lee, T., Shadbolt, N.: There’s gold to be mined from all our data. http://www.theodi.org/sites/default/files/Times%20OpEd%20TBL-NRS%20Final.pdf
3. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. Int. J. On Semantic Web Information System **5**(3), 1–22 (2009)
4. Open Source Initiative. The Open Source Definition. http://opensource.org/docs/osd
5. Decreto Legislativo 18 Agosto 2000, n. 267. Testo unico delle leggi sull’ordinamento degli enti locali. Gazzetta Ufficiale n. 227 del 28 Settembre 2000 – Supplemento n. 162

6. Direttiva 19 Dicembre 2003. Sviluppo ed utilizzazione dei programmi informatici da parte delle pubbliche amministrazioni. Gazzetta Ufficiale n. 31 del 7 Febbraio 2004
7. Open Cloud Manifesto. www.opencloudmanifesto.org/
8. del Bianco, V., Lavazza, L., Lenarduzzi, V., Morasca, S., Taibi, D., Tosi, D.: A study on OSS marketing and communication strategies. In: Hammouda, I., Lundell, B., Mikkonen, T., Scacchi, W. (eds.) OSS 2012. IFIP AICT, vol. 378, pp. 338–343. Springer, Heidelberg (2012)
9. Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: An empirical investigation of perceived reliability of open source java programs. In: 27th ACM Symp. on Applied Computing (SAC 2012), Riva del Garda, March 2012
10. Friso, C., Lenarduzzi, V., Taibi, D., Tosi, D.: How open source software products can support teaching in italian schools. In: 5th European Conf. on Information Management and Evaluation. Como., September 2011
11. Hauge, Ø., Ayala, C., Conradi, R.: Adoption of open source software in software-intensive organizations–A systematic literature review. Information and Software Technology **52**(11), 1133–1154 (2010)
12. Maldonado, E.: The process of introducing FLOSS in the PA: the case of Venezuela. Journal of the Association for Information Systems **11**(11), 756–783 (2010)
13. Rossi, B., Russo, B., Succi, G.: Adoption of FLOSS in public organizations: factors of impact. Information Technology & People **25**(2), 156–187 (2012)

The RISCOSS Platform for Risk Management in Open Source Software Adoption

X. Franch[1(✉)], R. Kenett[2], F. Mancinelli[3], A. Susi[4], D. Ameller[1], M.C. Annosi[5], R. Ben-Jacob[2], Y. Blumenfeld[2], O.H. Franco[1], D. Gross[4], L. Lopez[1], M. Morandini[4], M. Oriol[1], and A. Siena[4]

[1] Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
{franch,dameller,llopez,ohernan,moriol}@essi.upc.edu
[2] KPA, Raanana, Israel
{ron,ronb,yehudablu}@kpa-group.com
[3] XWiki, Paris, France
fabio.mancinelli@xwiki.com
[4] Fondazione Bruno Kessler (FBK), Trento, Italy
{susi,gross,morandini,siena}@fbk.eu
[5] TEI - Ericsson, Rome, Italy
mariacarmela.annosi@ericsson.com

Abstract. Managing risks related to OSS adoption is a must for organizations that need to smoothly integrate OSS-related practices in their development processes. Adequate tool support may pave the road to effective risk management and ensure the sustainability of such activity. In this paper, we present the RISCOSS platform for managing risks in OSS adoption. RISCOSS builds upon a highly configurable data model that allows customization to several types of scopes. It implements two different working modes: exploration, where the impact of decisions may be assessed before making them; and continuous assessment, where risk variables (and their possible consequences on business goals) are continuously monitored and reported to decision-makers. The blackboard-oriented architecture of the platform defines several interfaces for the identified techniques, allowing new techniques to be plugged in.

Keywords: Open source projects · Open source software · OSS · Open source adoption · Risk management · Software platform

1 Introduction

Risk management is a necessary and challenging task for organisations that adopt open source software (OSS) in their products and in their software development process [1][2]. Risk management in OSS adoption can benefit from the huge amounts of data that are publicly available about the adopted OSS components, as well as data that describes the behavior of OSS communities. The complexity and heterogeneity of the involved data sources, the need to integrate this data with contextual information related to the organisation and its ecosystem, and the convenience of combining different expertise involved in the assessment, call for adequate tools in support of all the

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 124–133, 2015.
DOI: 10.1007/978-3-319-17837-0_12

phases of risk assessment, from data gathering, to data statistical analysis, to the correlation of these data to the organisational strategic and business risks and assets.

In this paper, we present RISCOSS (www.riscoss.eu), a platform and related assessment methodology for managing risks in OSS adoption [3]. It defines several interfaces to a portfolio of identified measurement and risk management techniques, allowing new techniques to be plugged in if they implement these interfaces and follow well-documented protocols. RISCOSS builds upon a highly configurable data model that allows customization to several types of scopes to support different risk assessment perspectives. It implements two working modes: exploration, where the impact of decisions may be assessed in advance; and continuous assessment, where risk variables (and their consequences on business goals) are monitored, analysed and reported. This allows RISCOSS to support a holistic decision making process inside the adopter organisation.

2 Related Work

Several long-term projects, corporate programs and research initiatives have similar aims than the approach supported by RISCOSS. In Table 1 we summarize the characteristics of the most related European projects that propose methodological approaches and tool support for measuring several aspects of OSS projects, mainly to evaluate the quality of the OSS components and the communities behind them. In particular we refer to the projects with objectives:

- *FLOSSMetrics* (www.flossmetrics.org): constructing, publishing and analysing a large scale OSS database with metrics on OSS development;
- *QualiPSO* (qualipso.icmc.usp.br/OMM/): improving the quality & maturity of OSS projects;
- *QualOSS* (www.libresoft.es/research/projects/qualoss): defining a method to assess the robustness and evolvability of OSS projects;
- *ALERT* (www.alert-project.eu): improving the quality of the software acting on the overall bug resolution process in OSS collaborative environments;
- *OSSMETER* (www.ossmeter.com): supporting the process of discovering, comparing, assessing and monitoring the health, quality, and activity of OSS;
- *MARKOS (*www.markosproject.eu*)*: provides an integrated view on OSS projects, focusing on software functional, structural and license aspects.
- *SQO-OSS* (www.sqo-oss.org): proposing methods and a supporting platform for OSS code quality and community measurement and analysis.
- *U-QASAR* (www.uqasar.eu): providing a quality assurance methodology to assess the quality of software development projects for Internet applications.

All these approaches and platforms focus on the gathering and analysis of OSS data but they are not specifically oriented to inform about the risks derived from these data nor to suggest possible mitigation strategies at the technical and business level.

Table 1. European projects focusing on OSS data analysis

Project	Techniques	Knowledge Models	Tool support
FLOSS Metrics	Databases and analysis techniques to produce OSS project reports	Model of data to describe the characteristics of the different OSS projects	Tools to retrieve data from OSS repositories and produce statistics
QualiPSO	Integration of data from OSS repositories and statistical analysis	A software maturity model with three levels for projects categorization	A platform integrating tools to analyse the source code and the bug tracking systems
QualOSS	Checklist for data retrieving and statistical analysis	A quality model including characteristics, metrics and indicators	Tools to store checklist data and perform analysis of data
ALERT	Integration of data from OSS repositories; statistical analysis and recommendation techniques	Ontologies to support extraction and integration of different data sources	Components able to gather data from OSS sources and services for report visualization and recommendation
OSSMETER	Integration of data from OSS repositories and statistical analysis	Model for OSS forge description; models for the description of OSS quality attributes	Execution of project metrics; storage and analysis of the data and metrics
MARKOS	Integration of data from OSS repositories; license analysis	Ontologies to support the representation of concepts related to code and licenses	Tools to perform code analysis and license conflict analysis
SQO-OSS	Integration of data from OSS repositories and data analysis	Model for OSS quality based on source code and OSS community characteristics	Integration of metrics; analysis of the data through an array of algorithms
U-QASAR	Data gathering on the progress and quality of software development	Models describing software quality and contexts	Platform to obtain an objective value of the software development process quality

Companies and other organizations may also implement similar programs in their development cycles. For instance, we can refer to Codeface, an extensible platform developed by Siemens (http://siemens.github.io/codeface) that aims at gathering data from different OSS sources, to analyse them and to present them to the analyst in a configurable dashboard to support decision-making. The Black Duck suite (https://www.blackducksoftware.com) provides a set of tools for the automated governance and compliance of OSS across the application development lifecycle. From the quality assessment point of view, we can mention the Software Quality Assurance and Trustworthiness (SQuAT) programme at the OW2 consortium aimed at enhancing the perceived reliability of near 50 mature projects in the OW2 code base (http://www.ow2.org/view/About/SQuAT). Some approaches target specific aspects as license assessment, e.g. Palamida (http://www.palamida.com), which provides tools to verify if there is any intellectual property violation in a project adopting OSS; White Source (http://www.whitesourcesoftware.com) provides a solution for companies that need to manage their open source assets to ensure license compliance and reduce risk.

Several recent works face with mining and analysis of OSS projects mainly to assess or predict their quality. For example, in [4] the GHTorent project is presented that had the purpose of collecting data related to different aspects of the quality of the source code for all public projects available on Github. In the area of defect prediction for quality improvement, Peters et al. [5] introduce guidelines to be used in the building of software quality predictors in case of scarcity of data while D'Ambros et al. present a comparison between the different prediction approaches [6]. Zhang et al.

present in [7] a study for the specification of a universal defect predictor. Gamalielsson et al. [8] define the health of an open source ecosystem as an important decision factor when considering the adoption of an OSS component. In [9] the trustworthiness of OSS projects are predicted through the study of the Elementary Code Assessment. RISCOSS can benefit from these studies since it can integrate such quality models and techniques in the risk analysis platform. Adhering to the position defended by Noll et al. [10], RISCOSS calls for human-based qualitative analysis as a necessary component. Some authors define several risks that are associated with adopting an OSS component: project health [11], economic loss and adverse effects on the business processes of the organizations [12], the lack of effective and timely OSS community support for dealing with possible integration problems [13]. Hauge et al. [14] discuss several risks related to OSS adoption and identifies steps for reducing several of these risks. RISCOSS has a holistic perspective that integrates all of these elements in a platform to managing risks related to OSS adoption.

3 RISCOSS Main Functionalities

The main objective of the RISCOSS platform is that of facing the problem of managing risks in a holistic way, supporting the data gathering from the environment of the adopting organisation and from the organisation itself, the analysis of this data for the purpose of OSS risks identification and impact analysis, and the presentation of the data for decision making [3]. Moreover, the RISCOSS platform is designed to support two main operative modes. On the one side, it supports the analyst in performing an explorative analysis and assessment of the OSS ecosystem (in terms of communities and components), for example assessing the convenience of integrating an OSS component in the solution. On the other side, the platform implements a continuous assessment cycle that allows detecting the emerging risks related to OSS choices once, for example, an OSS component has been integrated inside a project.

Based on these basic requirements, some functionalities are proposed which are linked according to the workflow depicted in Fig. 1:

- *Set up of the risk management platform*. When an organization decides to adopt RISCOSS, the platform gathers the needed information to set up a risk management plan (in particular, to determine the key risk indicators), in order to configure the platform infrastructure to the organizational needs. This functionality allows initialising all the knowledge base of RISCOSS having it tailored for the particular organisational environment, including the representation of the business ecosystem where the organisation lives.
- *Identification of the risk assessment level and perspective* (*Elicit scope*). This use case offers the possibility to define a new scope of risk management (see Section 4). Every time one such scope is modified (remarkably, when it is created, e.g. a new project starts, a new OSS component is adopted), it is necessary to set up an organisational view and risk management resources and functionalities for it.
- *Risk assessment*. At any moment, the user may require explicitly risk assessment via situation inspection, what-if analysis by means of e.g. exploration of alternatives, deeper analysis of risk indicators [15], etc. Risk assessment may eventually end up with a change of scope in order to support a holistic risk detection.

- *Reaction to some key risk indicator violation*. As projects evolve and events occur, key risk indicators may be violated. RISCOSS monitors these violations and alerts are triggered when this happens. Then, risk analysis is performed to analyse and eventually solve these situations. Some of the events will be captured by the platform itself by measuring key risk indicators (e.g., a community may be detected to be inactive), some others must be communicated explicitly by decision makers, experts or analysts (e.g., some developer gets a relevant certification or training). Anyhow, this functionality allows for the evolution of the knowledge of the platform following the changes in the organisation and the related ecosystem.

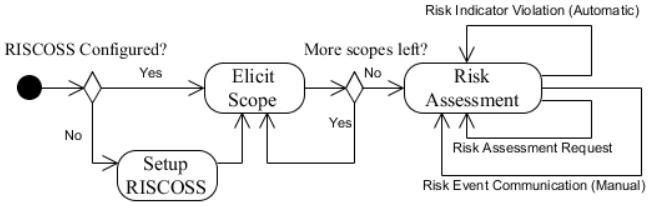

Fig. 1. Workflow for RISCOSS use cases

4 RISCOSS Scopes

We define *scope* as any unit of analysis that can be put under RISCOSS' control. Some organizations may want to monitor the full business; some others may just want to assess the risks related to the adoption of a particular OSS component. In the long term, our platform should be able to cover this entire spectrum. The concept of scope is important to structure the knowledge (and associated actions) managed in RISCOSS. If we refer to risks, every scope may have its different set of risks, e.g. coming from the adoption of some OSS strategy at several levels/scopes. For example, we can have risks as: losing current market position at the level of the organization; not delivering some release in time at the level of the product; involving more resources than expected at the level of the process; exceeding the allocated budget at the level of the project; incorrect selection at the level of the OSS component.

Fig. 2 shows a general view of the concept of scope, its relationships and its reifications, which are currently five (i.e., we have five types of predefined scopes). *Organizational unit*, that wants to supervise a complete portfolio; in a typical organisation, an organisational unit can be seen as a department. *Product*, a commercial good commercialized by the company; it does not necessarily have to be a software product, but of course needs to have some software part that is partially or totally open source. *Process (service)* such as, product manufacturing or product delivering. *Project*, for example, adding a new feature to the current release of a component, or making the necessary steps to deploy a bespoke component in an OSS community. *An OSS component* that is the finest-grained case, and an organization may be interested just in monitoring some adopted OSS component, belonging to a community.

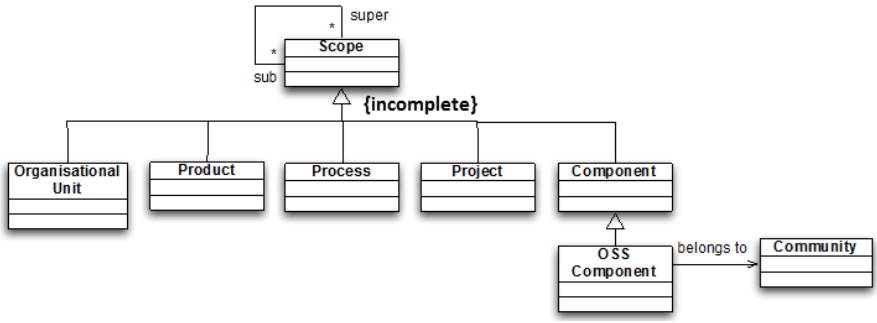

Fig. 2. General RISCOSS scope model

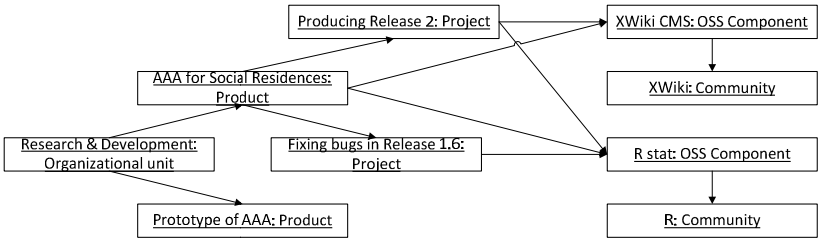

Fig. 3. An instantiation of the RISCOSS scope model

The class diagram shows how scopes are highly configurable, because: 1) the specialization is "incomplete", so that new types of scope may be added; 2) the reflexive association allows to build arbitrary hierarchies of scopes, so that in one organization, a project may include products, whilst in other a product may include projects.

In Fig. 3, we show an instantiation of the scope structure in which the Research & Development Unit of a Research institution has an OSS business strategy in place for the production of research prototypes and pre-competitive products to be tested on the field. The organization is focusing on two products: "Ambient Aware Assistance" (AAA), an environmental monitoring platform, and "AAA for Social Residences" (AAASR) that is a system for monitoring health conditions of patients living in social residences. The organization is currently running two projects in parallel for AAASR, "Producing release 2" and "Fixing bugs in release 1.6". The products are mixing different types of components, and for OSS components it adopts "XWiki CMS", an OSS web content management system developed by the "XWiki community", and "R stat", a complex statistical package, deployed by "R community". This second component is used only in the release 2. The Research institution would also like to affect "R community" releasing new statistical procedures to the community. Moreover, the organisation wants to restrict the scope of analysis of projects' components to those that are part of some commercialized product, i.e. if an OSS component is just used as part of the project management (e.g., a version control management tool) the organization is not interested in it.

5 Tool Architecture

The section introduces the basic RISCOSS platform architecture (see Fig. 4). The central element is a content management tool, XWiki, which is the unifying element

of the platform and covers three basic responsibilities: (i) Offering an interface to the user for dialoguing with the RISCOSS platform: as such, XWiki offers and organizes the required forms and dashboards, maintains documents, and supports, as a wiki tool, collaborative work as needed. (ii) Integrating other tools that perform functionalities required by the platform: XWiki is the umbrella that coordinates these tools, gathering the results of their computation to feed internal data structures, accessible from the tools in a blackboard architecture fashion, and allowing other tools to initialise their calculations. (iii) Accessing the reusable knowledge of the platform, namely the model patterns, data, form templates created by experts. The platform defines families of tools that are integrated into XWiki by means of well-established interfaces:

- *Questionnaire tool*. The tool gathers from decision-maker and experts of a given organisation the information related to the characteristics of the organisation and the initial scopes.
- *Goal and risk modelling tools*. The questionnaires are used, among other things, to create models that represent the ecosystem of the organization, with organizational goals stated using *i** [16], and risks models bound to them considering risks that are related to software quality, community behaviour or OSS licenses. This allows making explicit the consequences of risks in the OSS ecosystem.
- *Risk reasoning engines.* In particular:
  - *Logical reasoning tools.* These tools perform model analysis and reasoning in order to allow for risk identification and management. The reasoning tool currently implements model label propagation [17] techniques and disjunctive logic algorithms [18] in order to identify and mitigate the risks that can occur given specified environmental situations
  - *Statistical reasoning tools*. RISCOSS relies, among others, on the implementation of Bayesian Network based components that are used to reason about the correlation between the measures obtained by the analysis of the OSS communities and the strategic and business risks of the OSS adopters. These tools rely on a strong interaction with experts and analysts to allow for an assessment and revision of the correlations between the identified measures and the strategic and business risks.
- *Measuring and monitoring framework* composed of several Risk Data Collectors. These tools measure risk indicators and feed XWiki with the obtained values, and also monitor, detect and communicate anomalous situations from the risk management point of view. To do so, this framework needs to: 1) obtain data from the selected data sources (such as license or code quality analysis tools or blogs, forums and mailing lists of the communities); 2) implement basic statistical analysis, for example to compute statistical distributions of data [19], analysis of OSS communities structure and behaviour [20],
- *OSS Risk Dashboard.* This element collects the output of the risk assessment process to visualise it in a single view summarising the main aspects and giving the possibility to enter in the details of the single information. Here we envisage means to show the risk exposure of the organisation with respect to, for example, the adoption of a particular OSS component, or the result of a *what-if* analysis process.

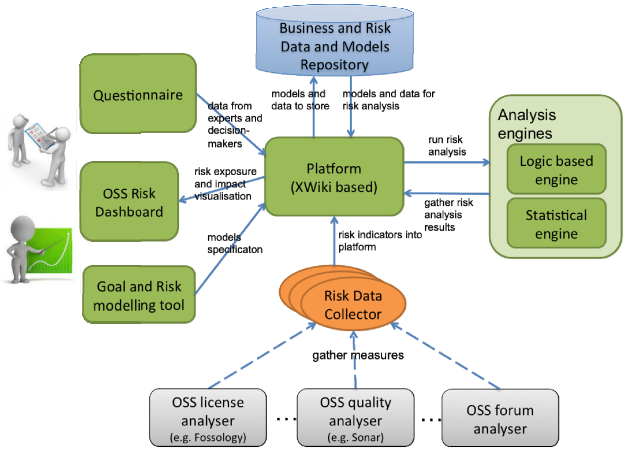

Fig. 4. RISCOSS platform logical view

In addition to these tools we can envision some other types to be added later in the platform, for instance, estimation cost tools to bind risks to project cost estimation.

It is worth to mention that XWiki manages a repository of reusable knowledge (the platform knowledge base). In this repository, we store the artifacts that can be reused for future and continuous assessment, such as patterns for goal and risk models.

Fig. 5 shows a component-oriented view that highlights the main type of components present in the RISCOSS platform and how they are interconnected.

The RISCOSSPlatform-XWiki component implements the RISCOSSPlatform API by using the APIs provided by XWiki. This component provides all the business logic for operating the RISCOSS Platform on top of the XWiki Platform. It requires some tools implementing the Tool API in order to perform the actual analysis. In particular these tools are the way for extending the RISCOSSPlatform with new capabilities, e.g. new risk analysis tools. A Tool component, on the other hand, provides the Tool API interface and requires the ToolConfigurationProvider API interface. This allows the tool to ask, in a standard way, configuration parameters that might be needed for its initialization. The RISCOSSPlatform-XWiki component is responsible to manage the persistence and retrieval of this information. This will be managed using the XWiki Platform that will store this information and will also provide a dedicated UI for manipulating it. The RISCOSSPlatform implementation is made available to the XWiki Platform via a component implementing the ScriptService API. This component will expose the relevant methods of the RISCOSSPlatform API to the UI component of the XWiki Platform that allows invoking the functionalities of the RISCOSS Platform in order to provide to the end user a user interface through which interact with it. The user interface will be also used to show the status and the results of the analysis performed, and the content of the collected information (i.e., the RISCOSS Knowledge base). The Storage component will provide the means for storing data persistently in a DBMS that supports the JDBC API.

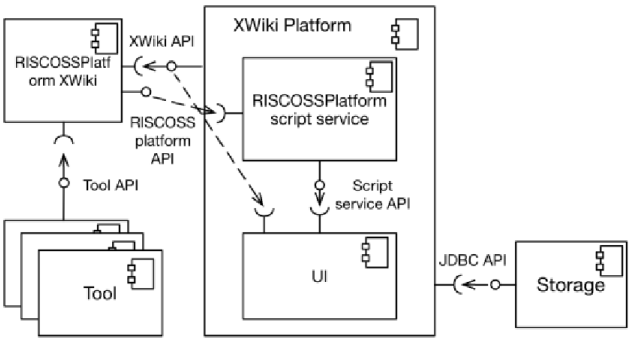

Fig. 5. RISCOSS platform deployment view

6 Discussion

The platform is currently in active evolution. The part already implemented consists of the set of main components for the different tool orchestration and a set of user interaction and risk and business analysis tools. The risks currently addressed in the platform are those related to OSS licenses violatoin and to software quality. The architecture is highly modular and allows for the definition of several possible tools that with the help of the coordination of the XWiki CMS platform can easily implement other services for the analyst. In particular, a possible direction of evolution is that of increasing the possibility of the decision-maker to interact with the reasoning engines via questionnaires and the implementation of new reasoning engines able to exploit this user knowledge in a more interactive way.

In the line of facilitating the use of the platform in different organizational settings from OSS communities to small and large companies adopting OSS, several deployment schemas have been considered, from a web-based installation where the different users that can create their own workspace and, at the same time use the common knowledge collected by the platform from the different users, to an organization specific installation, that can be also web-based, but that is confined in the premises of the organization. The first solution implicitly allows for another point of extension that is the creation of a large and evolving Business and Risk models knowledge base that can be reused and extended by other analysts, so promoting the creation of a community around the building of new knowledge about risks.

7 Conclusions

In the paper we presented the RISCOSS platform supporting risk assessment activities in OSS adoption. The peculiar aspect of the platform is that it aims at supporting the whole decision making process: from the gathering of the data from heterogeneous sources, to the risk indicator identification, to the modelling and assessment of the impact of the risks to the strategic and business goals of the organisation. We are currently deploying the platform in several contexts (large company with complex organizational structure, public administration, SME IT provider, and a community hosting ca. one hundred OSS projects) to improve its functionalities and knowledge.

Acknowledgement. This work is a result of the RISCOSS project, funded by the EC 7th Framework Programme FP7/2007-2013, agreement number 318249.

References

1. Li, J., et al.: A State-of-the-Practice Survey of Risk Management in Development with Off-the-Shelf Software Components. IEEE TSE **34**(2) (2008)
2. Hauge, Ø., Cruzes, D.S., Conradi, R., Velle, K.S., Skarpenes, T.A.: Risks and risk mitigation in open source software adoption: bridging the gap between literature and practice. In: Ågerfalk, P., Boldyreff, C., González-Barahona, J.M., Madey, G.R., Noll, J. (eds.) OSS 2010. IFIP AICT, vol. 319, pp. 105–118. Springer, Heidelberg (2010)
3. Franch, X., et al.: Managing risk in open source software adoption. In: ICSOFT 2013 (2013)
4. Gousios, G.: The GHTorent dataset and tool suite. In: MSR 2013 (2013)
5. Peters, F., Menzies, T., Marcus, A.: Better cross company defect prediction. In: MSR 2013 (2013)
6. D'Ambros, M., Lanza, M., Robbes, R.: An extensive comparison of bug prediction approaches. In: MSR 2010 (2010)
7. Zhang, F., Mockus, A., Keivanloo, I., Zou, Y.: Towards building a universal defect prediction model. In: MSR 2014 (2014)
8. Gamalielsson, J., Lundell, B., Lings, B.: The nagios community: an extended quantitative analysis. In: Ågerfalk, P., Boldyreff, C., González-Barahona, J.M., Madey, G.R., Noll, J. (eds.) OSS 2010. IFIP AICT, vol. 319, pp. 85–96. Springer, Heidelberg (2010)
9. Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: Predicting OSS trustworthiness on the basis of elementary code assessment. In: ESEM 2010 (2010)
10. Noll, J., Seichter, D., Beecham, S.: A qualitative method for mining open source software repositories. In: Hammouda, I., Lundell, B., Mikkonen, T., Scacchi, W. (eds.) OSS 2012. IFIP AICT, vol. 378, pp. 256–261. Springer, Heidelberg (2012)
11. Piggot, J., Amrit, C.: How healthy is my project? open source project attributes as indicators of success. In: Petrinja, E., Succi, G., El Ioini, N., Sillitti, A. (eds.) OSS 2013. IFIP AICT, vol. 404, pp. 30–44. Springer, Heidelberg (2013)
12. Petrinja, E., Sillitti, A., Succi, G.: Comparing OpenBRR, QSOS, and OMM assessment models. In: Ågerfalk, P., Boldyreff, C., González-Barahona, J.M., Madey, G.R., Noll, J. (eds.) OSS 2010. IFIP AICT, vol. 319, pp. 224–238. Springer, Heidelberg (2010)
13. Ayala, C., Cruzes, D.S., Nguyen, A.D., Conradi, R., Franch, X., Höst, M., Babar, M.A.: OSS integration issues and community support: an integrator perspective. In: Hammouda, I., Lundell, B., Mikkonen, T., Scacchi, W. (eds.) OSS 2012. IFIP AICT, vol. 378, pp. 129–143. Springer, Heidelberg (2012)
14. Ligaarden, O.S., Refsdal, A., Stolen, K.: ValidKI: a method for designing key indicators to monitor the fulfillment of business objectives. In: BUSTECH 2011 (2011)
15. Yu, E.S.K.: Modelling Strategic Relationships for Process Reengineering. PhD thesis, University of Toronto, Canada (1995)
16. Nilsson, N.J.: Problem-solving Methods in Artificial Intelligence. McGraw-Hill (1971)
17. Leone, N., et al.: The DLV System for Knowledge Representation and Reasoning. ACM Transactions on Computer Logic **7**(3) (2006)
18. Kenett, R.S., Zacks, S.: Modern Industrial Statistics: with applications in R, MINITAB and JMP, 2nd (edn.). John Wiley and Sons (2014). With contributions by D. Amberti
19. Salter-Townshend, M., White, A., Gollini, I., Murphy, T.B.: Review of Statistical Network Analysis: Models, Algorithms, and Software. Stat. Analysis and Data Mining **5**(4) (2012)

Intellectual Property and Legal Issues

First Results About Motivation and Impact of License Changes in Open Source Projects

Robert Viseur[1,2(✉)], and Gregorio Robles[3(✉)]

[1] CETIC, Rue des Frères Wright, 29/3, B-6041 Charleroi, Belgium
robert.viseur@cetic.be
[2] University of Mons (Faculty of Engineering), Rue de Houdain, 9, B-7000 Mons, Belgium
robert.viseur@umons.ac.be
[3] Universidad Rey Juan Carlos, 28943 Fuenlabrada, Spain
grex@gsyc.urjc.es

Abstract. Free and open source software is characterized by the freedoms and criteria that are warranted by specific licenses. These licenses describe the rights and duties of the licensors and licensees. However, a licensing change may be necessary in the life of an open source project to meet legal developments or to allow the implementation of new business models. This paper examines the motivations and impacts of license changes in open source projects. After a state of the art on the subject, a set of case studies where projects changed their license is presented. Then a set of motivations to change licenses, the ways to legally make this change, the problems caused by this change and a set of benefits of the license change are discussed.

Keywords: Open source · Intellectual property · License · Contributor agreement · Business model

1 Introduction

Licenses are thought to be selected at the beginning of the project with no posterior change (Fogel, 2005). They give the rules of the collaboration which everybody agrees to if participating in the project. To some extent, they provide a sort of "constitution" or legal agreement of how the project is developed and distributed. A change in the license is something very complex, not only because the aforementioned rules are changed, but also because it requires all contributors to agree on the license change.

One of the first, and most notorious license changes was performed by Mozilla in the late 90s (www-archive.mozilla.org). Due to incompatibility issues with the GPL noticed by the FSF, Mozilla decided to change from its dual NPL/MPL license to MPL/GPL/LGPL. This meant that all developers that had contributed to Mozilla had to explicitly give their consent to the change. Mozilla's website listed a number of contributors that could not be contacted or were missing. The license change finished when only around a dozen of missing confirmations from developers were still

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 137–145, 2015.
DOI: 10.1007/978-3-319-17837-0_13

required; their code was identified and rewritten or removed. Another famous relicensing project is KDE whose purpose was to solve some compatibility issues with GPLv3. The projects leaders were obliged to identify and get agreement from several hundreds of contributors to that major free and open source project.

Despite the importance of the issue for the governance of open source projects, there are few scientific papers dedicated to the motivations and the impact of licenses changes in open source projects. Our paper proposes to contribute to thinking on that issue. It is organized in four sections. The first one includes a brief state of the art about motivation and impact of the license changes. The second one is dedicated to the presentation of the methodology and the analysis of several case studies. The third one consists in a discussion of our first findings. The last one proposes some conclusions and perspectives.

2 Background

Free and open source projects (also said FOSS or FLOSS) are covered by specific licenses that warrant the free and open source nature of software as defined by the Free Software Foundation (fsf.org) and the Open Source Initiative (opensource.org). Those licenses can be divided in two main families (de Laat, 2005; St. Laurent, 2004; Viseur, 2013b). The first one includes the academic or permissive licenses; the second one, the copyleft, reciprocal or restrictive licenses. The permissive licenses allow the use of the source code in proprietary software. The Apache, BSD and MIT licenses are famous examples. Contrariwise, copyleft licenses impose limitations on licensees of any derivative work, such as the conservation of the license or the availability of the source code. The AGPL, CDDL, CPL/EPL, GPL, LGPL, MPL and OSL licenses are famous examples of copyleft licenses.

Ten years ago the copyleft licenses represented more or less 80% of all the open source projects. However the success of permissive licenses has grown over time. For example, Hofmann *et al.* (2013) noted that "*the combined effect of increased commercial investment with the need for competitively differentiated products built on top of that shared investment has lead to an increase of permissively licensed projects*".

We identified several authors working on the license changes and their impact on the open source projects. The studies are not numerous and are often recent.

The main study was processed by Di Penta *et al.* (2012). The authors identified several motivations for license changes such as the "*changes in the legal landscape, commercial code licensed as FOSS, or code reused from other FOSS systems" and the "evolution of a license per se*" (i.e. GPL v2 → GPL v3). They draw their conclusions from a set of examples and empirical study analyzing, including ArgoUML, Eclipse-JDT, FreeBSD and OpenBSD, Mozilla and Samba. They found a wide variety of licenses, even in source codes that are supposed to be homogeneous, due to a period of transition after the decision to change from a license to another or a version to another one.

Savola and Anttila (2012) published a seminar report dedicated to open source software license changes. They based their seminar report on examples from scientific literature including Di Penta *et al.* (2012). For each example they identified the motivations and the result of the license change. Their selection of license changes includes Netbeans IDE, MySQL, Mono and Java. The report identifies reasons for the license changes (including better defendability, more clear license text, gain more users, better suitability for commercial usage, more paying customers, prevent commercial usage without paying license fee, reduce viral effect, partner's requirement and wider distribution) together with the problems (including license incompatibilities and push-back from the community) and the benefits (including wider distribution, user concerns about copyright and higher money income) after the license change.

Santos *et al.* (2011) worked on projects that had changed their type of license over the years. Their final sample is of 756 free and open source software for a total of 1012 intellectual property policy (IPP) interventions (i.e. changes of license type). They studied the impact of the IPP intervention on FLOSS attractiveness. The authors explored the transitions. They calculated their frequency and their positive or negative impact on the project attractiveness. The results revealed a bias in favor of the confidential Academic Free License. Note that the impact of the license type of open source software success is always a debated topic (Viseur, 2013a).

Viseur (2013b) studied the evolution of business models of open source editors facing cloud computing. The study was focused on e-business software including Enterprise Resource Planning (ERP), Customer Relationship Management (CRM) and e-commerce software. The author showed evolutions in license choices. A first pattern is related to companies switching to dual licensing model, often with technical differentiation between community and commercial release of the software (e.g., Compiere). A second pattern is related to companies facing to the rise of business software in SaaS mode and switching to copyleft licenses with network effect (e.g., OpenERP). Broadly speaking, the copyleft licenses with network effect impose the option to download the source code if users communicate with the software interface. Famous examples are the GNU Affero General Public License (AGPL) and the Open Software License (OSL).

3 Methodology and Cases Studies

We list a set of project that encountered at least one license change (see Table 1). The list was manually fed by the authors. Some information collected in the context of a study about forks was also used (Viseur, 2012). The list was completed by new examples that the authors obtained after Twitter requests.

Table 1. Documentation of license changes

Project	License (from)	License (to)
Alfresco	GPL with FLOSS exception	LGPL
ath5k	BSD	BSD + BSD/GPL (fork)
Compiere	MPL	GPL + proprietary
Dimdim	MPL	GPL
ExtJS	LGPL-style	GPL + proprietary
IP Filter	IPFilter (BSD-Like)	IPFilter (modified)
Java	SCSL (hybrid)	GPL with classpath exception
Joomla (framework)	GPL	LGPL
JQuery	GPL/MIT	MIT
Mongoose	MIT	GPL + proprietary
Mono	GPL	MIT/X11
Mozilla	NPL MPL	MPL MPL/LGPL/GPL
MySQL	LGPL	GPL
OpenERP	GPL AGPL	AGPL LGPL
OpenStreetMap	CC-BY-SA	ODBl
SharpDevelop2	GPL	LGPL
Paint.Net	MIT	Proprietary
Squeak	Squeak Apple Public License Apache License	Apple Public License Apache License MIT/Apache License
SugarCRM	SPL (MPL-like) GPL	GPL + proprietary AGPL
Talend	GPL	LGPL
Trolltech	QPL GPL	GPL + proprietary LGPL
VLC	GPL	LGPL
WebM	WebM	WebM (modified)
XFree86	MIT	MIT with credit clause

We analyze each case by focusing on a set of criteria: the application domain, the original and the new license, the projects leaders' motivation to change, the way the license change was conducted and the impact on the open source project. Our work is mainly based on scientific or professional sources such as Bert-Erboul (2013), de Laat (2005), Di Penta and German (2010), Hamerly and Paquin (1999), Välimäki (2003), Viseur (2012), Viseur (2013a), Viseur (2013b) and Willis (2012). The issue of license change has usually not been the main subject of the scientific papers; however, professional sources document some aspects for specific projects. We describe below four representative cases: Mozilla, OpenERP, Trolltech and VLC.

Mozilla

Domain - The Mozilla Foundation develops several tools for the World Wide Web. The most famous are the discontinued Mozilla suite and the lighted Firefox Web browser.

License - The project was published in 1998 under NPL license. A conflict with the community led to the publication under the first version of MPL license. The LGPL and the GPL later appeared in the development (triple license scheme), in order to address problems of incompatibilities with third-party projects. The project also published an evolution of the MPL license that solved incompatibility issues with the GPL license.

Scope - The license changes impacted the whole project.

How - There is no copyright assignment. The license changes needed contributor agreement. The Mozilla Foundation asked for the agreement of committers by choosing the acceptable licenses.

Impact - Simplifying reuse in some third-party products is assumed.

OpenERP

Domain - OpenERP (now Odoo) is an open source Enterprise Resource Planning (ERP) software including Customer Relationship Management (CRM) features.

License - The software moved from GPL to AGPL in 2009 in order to avoid the use in SaaS mode without publishing the modifications. The software editor proposes its own SaaS offer based on OpenERP. In 2011 the company introduced a new license, an AGPL with Private Use License (dual licensing) allowing to keep the modifications private in case of SaaS use. A new license change is planned (from AGPL to LGPL).

Scope - The license change affects all the software except the OpenERP Web Client under MPL license.

How - The process of the license change is not clearly documented.

Impact - The OpenERP company evolved its business model and was able to respond to new market trends. However the second license change was associated with the emergence of tensions within the community.

Trolltech (Qt)

Domain - The Qt software is a cross-platform application framework.

License - The software was published under QPL license. Then a dual licensing scheme (QPL + GPL) was applied under the pressure of the KDE community that used Qt intensively. Finally, after Nokia bought Trolltech, the decision to publish the software under

the LGPL license in order to spread the technology was taken. The project is currently managed by another company, Digia, and "*some modules are available under LGPLv2.1, LGPLv3 or GPL v3 and some other modules under LGPLv3 only*" (qt.io).
Scope - The whole library was impacted by the successive license changes. However the actual release licensing scheme is heterogeneous.
How - The company was copyright owner.
Impact - The publication under GPL satisfied the community and the Harmony fork was abandoned. The impact of the publication under LGPL has not been quantified. The current license schema is complex.

VLC
Domain - The VLC project offered a cross-platform media player and a streaming media server. The project was initiated by students from Ecole Centrale de Paris.
License - The project has been covered by GPL v2 (or higher) since 2001. The license was changed for LGPL v2.1 (or higher) in 2011.
Scope - The license change was implemented for modules impacting third party vendors.
Motivations - There were several motivations, including "*making VLC compatible with various gadget-vendor application stores (e.g., Apple's)*" (Willis, 2012).
How - Two hundred and thirty contributors had to be contacted. It was difficult to contact them and get response. With 150 responses, the project owned an agreement for 99.99% of the code for libvlc and libVLCcore. It was more difficult for other parts, for which it was necessary to delete, reimplement, refactor into separate files or drop some code. The code was kept under GPL if there was no impact on third party vendors.
Impact - Some modules had to be reimplemented (less than 5). The impact on Apple is unknown (opacity).

4 Discussion

Beyond the four cases that we have presented so far, we identified a set of motivations for license changes (see Table 2):

1. The license can be changed for compatibility. The open source project can thus be distributed in new packages (e.g., Java), be integrated in wider source code (e.g., WebM) or reduce the license impact on third party add-ons development (e.g., Odoo, VLC or SharpDevelop2).
2. The license can be changed by the editor with the hope that the revenues will increase. This is the case when an editor organizes a transition to a dual licensing scheme (e.g., MySQL).
3. The license can be changed from the GPL license to a more permissive one in order to avoid how "derived work" is defined (e.g., Mono).
4. The license change can be motivated by the simplification of a multiple licensing scheme (e.g., JQuery).
5. The license change can be motivated by the desire to improve the relationships to community and limit the risk of fork (e.g., Trolltech).

6. The license change can be the consequence of an ego-conflict. That reason is often alleged but not proofed (e.g., IP Filter).
7. The license change can be an adaptation to evolutions in the competitive environment. For example, a lot of open source projects are preparing themselves for the cloud computing rise (e.g., OpenERP, now Odoo, or SugarCRM).
8. The license change can be require by a commercial partner (e.g., Mono).
9. The license change can be motivated by legal arguments. It can be to clarify the original license (e.g., IP Filter) or to adopt a more suitable license (e.g., OpenStreetMap).

Table 2. Motivations (the cumulated percentage must be higher than 100% as a license change can be associated to several motivations)

Expected benefit	**Case study** (#24)	**Percentage**
#1 Compatibility issue	Alfresco, Java, Joomla, Mono, SharpDevelop2, Squeak, Talend, Trolltech, VLC, WebM	41.7%
#2 Revenue growth	Compiere, ExtJS, Mongoose, MySQL, OpenERP, XFree86	25%
#3 Permissiveness	jQuery, Mono	8.3%
#4 Simplification of license scheme	jQuery	4.2%
#5 Relationships to community	Trolltech	4.2%
#6 Ego-conflict	ath5k, IP Filter, Paint.Net	12.5%
#7 Adaptation to competitive environment	OpenERP, SugarCRM	8.3%
#8 Partner requirement	Dimdim, Mono	8.3%
#9 Legal argument	IP Filter, OpenStreetMap	8.3%

The GPL license is often in the center of the license changes; either the project changes to the GPL for financial or compatibility reason, or the project moves away from GPL to a more permissive one in order to avoid the side effects of the GPL. Note that a license change can also be the result of several motivations at once.

The license change can be hindered by the shared property (among all contributors) of the source code. The issues can be managed in two ways:

1. The editor ensures that he owns the copyright on the source code (by contributors' agreement or by rewriting of contributions) and can change the license. This is the case of MySQL or Qt.
2. The responsible must get agreement from all the contributors (if necessary, by contacting them one by one). This has been the case for Mozilla or VLC.

We identified three problems that the license change can cause:

1. The project license can impact strongly or become incompatible with other projects (e.g., MySQL with PHP). The incompatibilities are often caused by copyleft licenses such as the GPL. An exception clause (e.g., classpath exception, FLOSS or FOSS exception, etc.) allows to reduce the virality effect of the license.
2. The new license, if it is permissive, can allow the appropriation of the source code by the editor, and may be undesired for community members (e.g., Mono).
3. The license change can irritate the community and lead to a fork (e.g., ath5k).

We identified three benefits the license change can bring:

1. The updated project can gain a wider distribution due to the license change (e.g., Squeak or Java). However, the updated project can contrariwise be affected by a smaller distribution due to the license change (e.g., IP Filter).
2. The editor who updated the license can gain more revenue (e.g., MySQL). However there is insufficient data to objectify the effect of license change on the company profitability (e.g., Mono).
3. The license change can lead to the abandonment of a fork (e.g., Trolltech Qt and Harmony) if it is part of community claims.

The case studies showed that the issue of license change was not specific to software. Indeed OpenStreetMap (openstreetmap.org) shows a similar problem in the field of databases. The ODBL license (opendatacommons.org/licenses/odbl/) was adopted because it was considered more suitable compared to CC-BY-SA license (creativecommons.org).

5 Conclusion and Perspectives

As preliminary results, our paper allowed to identify nine motivations, three negative impacts and three positive impacts for license changes.

We intend to continue this study with some future research. First, we plan to add new cases of license changes. For example: KDE (relicensing to be compatible with GPL v3), Samba (adoption of GPL v3) and Wikipedia (move to CC-BY-SA). The discovery of new cases might be facilitated by the use of free and open source projects data. For example, the FLOSSmole datasets (flossmole.org) contain license information (see database schema) that would be useful to automate the search of new interesting cases. Second, we want to study in detail the identified cases. In particular we found there was often a lack of documentation about the license change procedure (even for well known projects such as Qt), and that the impact was rarely objectified. We would like to enrich the cases with facts and figures allowing to better quantify the impact of license changes. So we wish to obtain a deeper understanding of the phenomena. Our final goal is to propose guidelines to open source project managers and entrepreneurs since there is already for license selection.

References

1. Bert-Erboul, C.: VideoLan: un cas d'école dans l'industrie informatique. Revue Terminal, pp. 117-133 (2013)
2. de Laat, P.B.: Copyright or copyleft? An analysis of property regimes for software development. Research Policy **34**, 1511–1532 (2005)
3. Di Penta, M., et al.: An exploratory study of the evolution of software licensing. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, vol. 1. ACM (2010)
4. Fogel, K.: Producing open source software: How to run a successful free software project. O'Reilly Media, Inc. (2005)
5. KDE, Projects/KDE Relicensing. https://techbase.kde.org/Projects/KDE_Relicensing (read: January 17, 2015)
6. Hamerly, J., Paquin, T.: Freeing the source: the story of mozilla. In: Open Sources: Voices from the Open Source Revolution. O'Reilly, January 1999
7. Hofmann, G., Riehle, D., Kolassa, C., Mauerer, W.: A dual model of open source license growth. In: Petrinja, E., Succi, G., El Ioini, N., Sillitti, A. (eds.) OSS 2013. IFIP AICT, vol. 404, pp. 245–256. Springer, Heidelberg (2013)
8. Mozilla, Mozilla Relicensing FAQ - Version 1.1. http://www-archive.mozilla.org/MPL/relicensing-faq.html (read: January 14, 2015)
9. Santos Jr., C.D., et al.: Intellectual property policy and attractiveness: a longitudinal study of free and open source software projects. In: Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work. ACM (2011)
10. Savola, S., Anttila, O.: Open Source Software License Changes, Open Source Software Development Seminars (November 22, 2012). https://wiki.oulu.fi/download/attachments/28092087/ossd_2012_savola_anttila.pdf (read: January 15, 2015)
11. St. Laurent, A.M.: Understanding Open Source and Free Software Licensing. O'Reilly Media (2004)
12. Välimäki, M.: Dual licensing in open source software industry. Systèmes d'Information et Management **8**(1), 63–75 (2003)
13. Viseur, R.: Fork impacts and motivations in free open source projects. International Journal of Advanced Computer Science and Applications **3**, 117–122 (2012)
14. Viseur, R.: Evolution des stratégies et modèles d'affaires des éditeurs Open Source face au Cloud computing. Revue Terminal 113-114
15. Viseur, R.: Identifying success factors for the mozilla project. In: Petrinja, E., Succi, G., El Ioini, N., Sillitti, A. (eds.) OSS 2013. IFIP AICT, vol. 404, pp. 45–60. Springer, Heidelberg (2013)
16. Willis, N.: Relicensing VLC from GPL to LGPL, LWN.net (November 21, 2012). http://lwn.net/Articles/525718/ (read: January 15, 2015)

On the Variability of the BSD and MIT Licenses

Trevor Maryka, Daniel M. German, and Germán Poo-Caamaño(✉)

University of Victoria, Victoria, Canada
{tsmaryka,dmg,gpoo}@uvic.ca

Abstract. The MIT/X11 and the BSD are two of the most important family of Free and Open Source (FOSS) licenses. Because these licenses are to be inserted into the files that use it, and because they are expected to be changed by those who use them, their text has suffered alterations over time. Some of this variability is the result of licenses containing template fields which allow the license to be customized to include information such as the copyright holder name. Other variability can be attributed to changes in spelling, punctuation, and adding or removing conditions. This study empirically evaluated the extent that the BSD and MIT/X11 family of licenses are varied, and the manner and frequency in which license texts vary from the original definition. The study found that the BSD family has little variability, with a significant proportion fitting the common standard. The MIT/X11 family of licenses exhibited significantly more variation, with a higher propensity to customize the license text. In addition, the MIT/X11 license has spawned several specialized variants which likely constitute different legal meanings. Based on these findings, recommendations are proposed on what variability needs to be accommodated by the Software Package Data Exchange (SPDX) which is in the process of standardizing the allowed variability of both licenses.

1 Introduction

There exist many open sources licenses. The Open Source Initiate[1] (OSI), responsible for the definition of open source, has approved 70 licenses[2] The Software Package Data eXchange[3] (SPDX), a consortium of non-profit and profit organizations that attempt to standardize licensing information across parties lists 306 different licenses[4].

Some of the most important open source licenses are the family of BSD licenses, and the family of MIT licenses. These licenses comprise a very large portion of open source licensed software; in a study it was found that 9.1% of Debian applications were licensed under BSD or MIT licenses [2]. Furthermore,

[1] http://osi.org
[2] http://opensource.org/licenses/alphabetical
[3] https://spdx.org
[4] https://spdx.org/licenses

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 146–156, 2015.
DOI: 10.1007/978-3-319-17837-0_14

these licenses, which are known as Academic [6] are of particular interest because they allow unlimited use the software with very few restrictions[5].

Both the BSD and MIT licenses are template licenses, because they have to be modified by the licensee to properly use them. The expected way these licenses are used is, first, by embedding the text of the license into each file; and second, by modifying the name of the copyright owner and related information in the file. Unfortunately, users of the license often further modify the text of license. It is a common practice of users of these licenses to replace some generic references to the copyright holders with (e.g. "the name of the organization" replaced with the name of the owner "ACME LTD"). In fact, the modifications made by users to the original BSD 4-Clauses license resulted into the BSD 3-Clauses and BSD 2-Clauses license (by dropping clauses from the original license).

This paper describes a empirical study of how the BSD and MIT licenses have been modified by its users. The contributions of this study include:: *a*) An empirical study documenting how the MIT and BSD are modified in practice in source code of Debian software packages; *b*) Analysis of this variability., and *c*) Recommendations for the SPDX Group on how to address the variability of these licenses in their future templates.. These results are being used by the SPDX Group to improve the templates of these licenses to match the way they are used in practice (without altering their legal meaning) and developers of tools that perform license identification.

2 Background and Related Work

The Software Package Description Group is a consortium of for-profit and non-for-profit companies created under the auspices of the Linux Foundation. One of its primary objectives is the creation of the SPDX Standard [5,7]. The intent of the standard is to help in documenting and exchanging the license and copyright information of software components. The current version of the standard (v1.2) describes an easily parsable format to use to document what files are part of a component, their licenses and copyright owners, and the effective license of the component [5,7]. The SPDX Standard is expected to facilitate the exchange of this information and the creation of tools for software license compliance around it. This requires clear guidelines of how to identify and document licenses, specially Free and Open Source (FOSS) licenses. While most research has concentrated on the licenses documented by the Open Source Initiative (the so called 70 OSI-approved licenses), there exist many more licenses. As mentioned above, SPDX currently lists 306 licenses and has a mechanism to submit new licenses for consideration[6].

License identification of source code is challenging [2,3]. In FOSS each file is expected to contain a comment (usually at the top) that documents how it

[5] The original BSD 4-clause license is an exception to this rule, unless the copyright owner is the University of California; for details, see https://spdx.org/licenses/BSD-4-Clause-UC

[6] http://spdx.org/spdx-license-list/request-new-license

is licensed. We will refer to it as the *license statement* of the file. The license statement of a file is expected to include who the copyright owner is and how the file is licensed. There are two methods in which a file documents its license: by-inclusion and by-reference. By-inclusion refers to a license that it used by including its text in the license statement of the file. For example, files that are licensed using the BSD and MIT licenses usually include the full text of the license in their license statements.

On the other hand, by-reference corresponds to licenses that are not included in the license statement of the file; instead, a link to the license is provided. For example, the Apache-2.0[7] is expected to be used by-reference; the Appendix *How to apply the Apache License to your work* indicates how to add such reference (e.g. by including the following text: *Licensed under the Apache License, Version 2.0 (the "License"); [...] You may obtain a copy of the License at: http://www.apache.org/licenses/LICENSE-2.0*).

Many licenses that are used by-inclusion are expected to be customized by their users. For example, the original BSD license (BSD-4-Clause-UC[8]—which has the Regents of the University of California as its copyright owner) was converted into a template license where the text of the copyright owner is expected to be filled by the user (thus becoming the BSD-4-Clause). For example, the following paragraph illustrates the variable text (between {}, bold use for emphasis) that should be replaced when the license is used[9].

> IN NO EVENT SHALL **{{COPYRIGHT HOLDER}}** BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES

By-inclusion template licenses should be modified only in the form indicated. Unfortunately it is often the case that users of the licenses modify the text further. As described in [2], it was found that licensors often change the spelling of licenses (English vs British), change grammar (add or remove punctuation to clarify intent), and in some cases, add or remove extra clauses of known licenses to create new licenses.

Some of these changes result in licenses that are so frequently used that they deserve to receive a name. As mentioned above, the original BSD-4-Clause-UC was converted to a template license and called BSD-4-Clause (the user must *fill* the template with the name of the corresponding copyright owner, replacing the "Regents of the University of California" found in the original BSD-4-Clause-UC). The BSD-4-Clause is shown in Figure 1 (as documented by SPDX).

In other cases clauses were removed, resulting into the BSD-3-Clause and the BSD-2-Clause. In the case of the BSD-2-Clause, when the copyright owner is the NetBSD Foundation, Inc. the license is known as BSD-2-Clause-NetBSD (this is no longer a template license). Apache-1.0 is a derivative of the BSD-4-Clause: it reused its four clauses and the liability and warranty terms; furthermore,

[7] In this paper we refer to FOSS licenses by their the SPDX standardized names.

[8] For the remaining of this paper, we will refer to licenses by their SPDX name, see spdx.org/licenses/

[9] http://spdx.org/licenses/BSD-4-Clause

```
1  Copyright (c) <year>, <copyright holder>
2  All rights reserved.
3  Redistribution and use in source and binary forms, with or without modification,
4  are permitted provided that the following conditions are met:
5  1) Redistributions of source code must retain the above copyright notice, this
6     list of conditions and the following disclaimer.
7  2) Redistributions in binary form must reproduce the above copyright notice, this
8     list of conditions and the following disclaimer in the documentation and/or
9      other materials provided with the distribution.
10 3) All advertising materials mentioning features or use of this software must
11    display the following acknowledgement:
12      This product includes software developed by the organization.
13 4) Neither the name of the organization nor the names of its contributors may be
14    used to endorse or promote products derived from this software without specific
15    prior written permission.
16 THIS SOFTWARE IS PROVIDED BY COPYRIGHT HOLDER ''AS IS'' AND ANY EXPRESS OR IMPLIED
17 WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
18 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT
19 SHALL COPYRIGHT HOLDER BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
20 SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
21 PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
22 BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
23 CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
24 ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
25 OF THE POSSIBILITY OF SUCH DAMAGE.
```

Fig. 1. SPDX specification of the BSD-4 Clauses. The variable sections are depicted in bold.

among many other changes, the Apache-1.0 fixed a spelling mistake of the BSD-4-Clause ("EXPRESSED" instead of "EXPRESS") and added "ITS" in "OR ITS CONTRIBUTORS".

These types of license customizations have created a problem for SPDX. The goal of SPDX is to document licenses and their use. This includes documenting the templates and how they are expected to be modified by their users. Unfortunately the original SPDX templates of these licenses have used different ways to document this variability. For example, SPDX documents the BSD-4-Clause as a template license, as shown in Figure 1. At it can be seen SPDX uses both <> and {{}} to document the sections that must be modified (shown in bold). Even though a template might not allow, a license is frequently modified. For example, in the BSD licenses it is common to see *COPYRIGHT HOLDER* replaced with the actual name of the copyright holder (line 16 in Figure 1.

In SPDX Version 1.2 this variability has been accounted for via a template language. It documents the variable sections of a license using regular expressions. It indicates where and how a license can be modified and still be considered the same license. SPDX is now in the stage of updating the templates for these licenses[10]. The challenge is not longer how to document the variability, but what variability should be documented as permitted without changing the meaning of the license.

[10] See https://github.com/dmgerman/spdxTemplates for the current state of these templates (not yet approved by SPDX); for example the template for the BSD-4-Clause can be found at https://github.com/dmgerman/spdxTemplates/blob/master/bsd-4-clause/bsd-4-clause.txt

3 Research Questions

The two most common template licenses in open source are the family of BSD licenses (BSD-4-Clause, BSD-3-Clause, and BSD-2-Clause, etc.) and the family of MIT licenses (MIT, X11, MIT-CMU, MIT-advertising, etc). In Debian 5.0 these licenses account for more than 5% of all the files, and more than 8% of the applications that in which all their source files were licensed under the same license [2].

This study attempts to answer two fundamental questions for these two families of licenses. Firstly, what is the degree of variation of a license? In other words, if we imagine the original license text as the root of a tree, what branches of variation in the text have been created? Secondly, what is the frequency that variation occurs at? In particular, can we find the common patterns of variability used, and separate these from the less commonly used variants?

4 Methodology

The subject of this study is the Debian Linux Distribution, version 6. Ninka was run on all the files of this distribution, encompassing more than 1.3 million source files contained in 10,014 projects. Ninka identified 42,653 licenses in the BSD family, and 28,205 licenses in the MIT/X11 family. The variability analysis of these licenses worked in a top-down manner, by first identifying the largest, most commonly used variations, working down to smaller, less commonly used variations to the point where further analysis of variability would be negligible.

Ninka uses a pipeline architecture, and divides license identification into several stages. Two of them were of value to this research: sentence matching, and license matching. Sentence matching corresponds to the process of matching a given sentence to a known valid licensing sentence (irrespective of the license it belongs to). Even though the entire license might not match a known license, one of more of its sentences might. This allows to identify sentences of BSD and MIT licenses in cases where the entire license didn't match a known license. When Ninka matches a sentence, it outputs the known sentence, and what (if any) known variability it exhibited.

Enumeration and analysis of variability was done in a top down manner, in three levels. Initially the exact license that Ninka identified, such as the strict SPDX BSD2 vs. the less strict BSD2 (as named by Ninka), was enumerated. This level will be designated "license level variability". Secondly, the variability of the construction of sentence tokens that comprise a license was enumerated. An example of this would be a non-SPDX license, such as BSD2, must have one or more individual sentences that are not the strict version. Multiple combinations of strict and non-strict sentences are possible. The combination of sentence tokens can be referred to as the "token signature", and this level of variability will be designated "token signature variability". Finally, the content of each variable section within a sentence token was extracted from the sentence token file and enumerated. This level or variability will be designate

5 Results

5.1 The BSD Family

We use a top-down approach to describe the variability of the BSD family of licenses. Figure 2 depicts this variability.

BSD-4. Ninka identified 3,251 licenses in Debian 6 as variants of the BSD-4-Clauses (8% of all BSDs). Of them, 1,887 licenses were identified as exactly the SPDX BSD-4-Clause. The rest 1,370 showed mostly small variations. The most common of these changes (1,179–86%) were due to changes in the non-endorsement clause statement (Lines 13-15 in Figure 1): 457 licenses replaced "nor the names of its contributors" with specific names (e.g. "the University nor of the Laboratory") or remove it altogether. Another 220 licenses alter alter the text "contributors" to "co contributors" or "co-contributors". A further 398 licenses had the "Neither" removed from this clause. A very small number licenses (75) modified the Clause-3 by replacing the word "by" with "for" (Lines 11-13 in Figure 1). All other variability of the BSD-4 license were negligible.

BSD-3. Ninka identified 22,577 licenses as the SPDX BSD-3-Clauses (this was the most common). A further 4,822 had small changes that can be considered still within the spirit of the BSD. As with the BSD-4, most changes were due to to variations in the Non-Endorsement clause (Clause 3): 4,579 (91%). In 3,702 "Neither" was removed (in 96% of these cases the "contributors" were removed too). Another 887 licenses had the text "its contributors" changed to "his contributors", "other contributors", or "any contributors". The remaining 61 licenses indicate specific contributors (as the BSD-4 mentioned above, e.g. "the University nor of the Laboratory"). The rest of the variants were negligible. Also, 887 (18%) non-endorsement clause changes are due to a change the text "nor the names of its contributors". 826 (93%) of these use some small variant on the qualifier before "contributors" such as "his contributors", "other contributors", or "any contributors". The remaining 61 cite specific contributors ("the University nor of the Laboratory"). All other variability of the BSD-3 license was ignored because its incidence represented less than 2%.

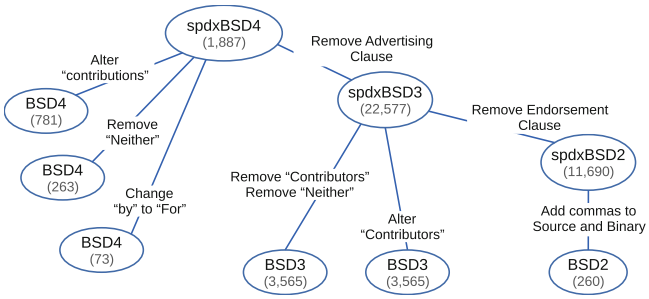

Fig. 2. BSD Variability Tree

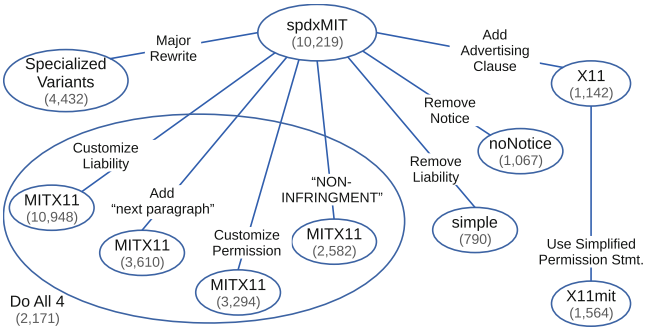

Fig. 3. MIT/X11 Variability Tree

BSD-2. Ninka identified 11,690 licenses as SPDX BSD-2. We found very little variability in licenses that can still be considered BSD-2 (only 458 licenses). 295 had variability in the Redistributions of Source Condition and 260 in the Redistributions of Binaries Condition. In most cases the changes are minor: 287 added a comma after the word "conditions", 268 added "above" before "copyright notice" (in both conditions). The remaining changes were negligible.

5.2 The MIT/X11 Family

The MIT/X11 family of licenses is considerably more fragmented the BSD family. Ninka divides the MIT/X11 family into eleven separate licenses, compared to the 3 documented by SPDX.

The most frequent license identified by Ninka is the MIT as documented by SPDX: 10,219 (36% of all MIT/X11 family). The tree of MIT/X11's significant variability, and the frequency of each variant is show in Figure 2.

Of the 17,986 (64%) licenses that are not the strict SPDX version, Ninka identifies five types of licenses which are closely derived from the SPDX definition: X11, X11mit, MITX11, MITX11noNotice, and MITX11simple. These non-specialized licenses account for 13,554 (75%) non-strict licenses.

Ninka also identifies five types of licenses which are much more specialized variants of the MIT/X11 license: MITold, BindMITX11Var, MIToldNoSellNoDoc, X11Festival, MITNoSellNoDocBSDvar (these are names that Ninka has given to each of these variants), and the SPDX MITNFA. 4,432 (25%) of the non-SPDX licenses are identified in this specialized category, and each license arguably represents a more significant re-writing of the strict license, rather than a smaller variation (with MITold likely being the a predecessor rather than a re-writing).

The non-specialized variant that is most closely derived from the SPDX version is the license Ninka identifies as MITX11. This license comprises 8,991 (66%) of the non-SPDX, non-specialized licenses. These MITX11 licenses are similar to the SPDX standard, but with at least one sentence changed.

Ninka identifies 1,142 (8%) licenses as X11 (Ninka's X11 is different from SPDX X11—the latter is not a template license and can only be used by the X11

Consortium). This X11 license is the SPDX MIT with an additional advertising material clause:

> Except as contained in this notice, the name of the **{{author}}** shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from **{{author}}**.

Ninka's X11mit accounts for 1,564 (12%) licenses and it is a variation of Ninka's X11 license with an old style, simplified permission statement:

> Permission to use, copy, modify, distribute, and sell this software and its documentation for any purpose is hereby granted without fee, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

MITX11noNotice accounts for 1,067 (8%) licenses, and corresponds to the SPDX MIT license with the notice statement removed:

> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

MITX11simple accounts for 790 (6%) licenses and is a variant of the SPDX MIT with the liability statement removed:

> IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE [...]

Variability in Sentences. Within the MIT family of licenses, we found that many variants shared the same type of variability at the sentence level.

Liability Statement. The most commonly varied statement is the liability disclaimer sentence. 10,948 (80%) of the non-strict, non-specialized licenses alter the text "AUTHORS OR COPYRIGHT HOLDERS" to either cite specific organizations/ authors (67%), or to vary it in some fashion (33%), such as changing it to "THE ABOVE COPYRIGHT HOLDERS".

Notice Statement. The second most commonly variant is the notice statement. 3,610 (26%) of the non-strict, non-specialized licenses vary the text of the notice statement:

> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

These licenses add the text "(including the next paragraph)" after the text "permission notice", implying that the warranty statement must also be included.

Permission Statement. The third most common variant is in the permission statement. 3,294 (24%) of the non-strict, non-specialized licenses vary the permission statement *Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"),...* in the following manner: 2,445 (74%) of licenses split "sublicense" to "sub license"; 655 (20%) licenses add an "on" before "limitation"; 582 (18%) licenses add a ", distribute without modifications" after "distribute".

"As Is" Statement. Finally, the fourth most commonly varied statement in the MIT/X11 family is the "As Is" warranty disclaimer sentence. This variation simply changes the word "NONINFRINGMENT" to "NON-INFRIGNMENT" (adding a dash). 2,852 (21%) of non-specialized, non-strict licenses add this dash. A significant proportion (2,171) of non-strict, non-specialized licenses use the non-strict versions of the permission, notice, "As Is", and liability statements at the same time.

6 Analysis

The SPDX version of each BSD license is exactly equivalent to the BSD licenses found in Debian 6 for the vast majority (85%) of licensing statements. The most commonly changed sentence is the non-endorsement clause found in BSD-3 and BSD-4. It is likely that this statement is most commonly varied because it contains a template field. The requirement to customize the template field likely increases the propensity to make other changes to the sentence, such as the removal of the word "Neither" and the customization of the "nor the names of its contributors" phrase.

The MIT/X11 family of license exhibits significantly more variability, with the majority of licenses not exactly matching the SPDX version of the license. The most significant and common variation of the license family is for the licensor to treat the text "AUTHORS OR COPYRIGHT HOLDERS" as a template field, and to customize it. Licensors also add conditions and grants such as "(including the next paragraph)", or "distribute without modifications", which may or may not alter the legal meaning of the license. Additionally, licensors adjust the spelling of words such as "sublicense" and "noninfringment", by adding dashes or spaces.

7 Recommendations

The BSD family of licenses very closely matches the SPDX standard, so little if any changes are required to the SPDX license list to accommodate its variability. If the word "Neither" was considered optional in the non-endorsement clause, an additional 4,100 Debian 6 licenses could directly match the SPDX, but that only represents 1% of the BSD family so this is likely a negligible variation.

The MIT/X11 family of licenses has significantly more variability, so small changes to accommodate potentially varied license texts would be beneficial to the SPDX standard, as it would increase the number of licenses that can accurately be included by users of the standard.

MIT/X11 Template. The most significant variation that SPDX could accommodate within the MIT/X11 family would be the alteration of the text "AUTHORS OR COPYRIGHT HOLDERS", in the liability statement, to be the template field "{{authors or copyright holders}}". This text has been treated as a de facto customizable field by licensors, despite the fact that this text was not intended to be customized. Accommodating this customizable field would increase the number of SPDX equivalent licenses in Debian 6 by at least 4,110, which is a 40% increase.

Equivalent Phrases. Much like the set of equivalent phrases used in the text normalization phase of Ninka, SPDX has published a small list of spellings which are considered legally equivalent [7]. The following recommended equivalent sets should be added to this list to accommodate variability found in BSD and MIT/X11 licenses:

- *"contributors"*, *"co-contributors"*, *"co contributors"*
- *"sublicense"*, *"sub-license"*, *"sub license"*
- *"noninfringement"*, *"non-infringement"*

The inclusion of these equivalent phrases would accommodate 5,517 occurrences of variability found. Note that this figure is greater than the number of individual licenses that can be accommodated, as the changes to "sublicense" and "noninfringment" may not be mutually exclusive.

MIT/X11 Additions. Adding the grant "distribute without modifications" to the list of permissions granted by the license may be legally redundant. If the licensee is permitted to redistribute a modified copy of software, then theoretically the licensee could redistribute a modified copy which is functionally equivalent to the original. As such, distributing the functionally equivalent modified copy may be equivalent to distributing the software without modifications, so the additional grant may be redundant. This implies that adding this clause constitute a license error and should be removed.

Derived Licenses. Some MIT/X11 licenses, like the specialized variants Ninka recognizes, and the variants that add the advertising clause, remove the notice clause, or replace the permission statement represent different legal meanings from the strict SPDX MIT license. These licenses need to be individually added to the SPDX license list to be accommodated by the standard. The process to add a license to the list is expensive [1], so these should be accommodated in a top down manner, with more frequently proposed/requested licenses added first. However, this is not currently the case. Licenses are currently considered for addition in the order in which they are proposed.

8 Threats to Validity

We identify the following threats to validity of this empirical study. Regarding External validity. The use of Debian 6 as a data source is likely representative of the BSD and MIT/X11 license families, as Debian offers a diverse and comprehensive view of FOSS landscape [4]. Additionally, the accuracy of Ninka would play a significant role in the external validity of this study, since Ninka has been externally verified to have a high accuracy of 96.6% [2].

Regarding Reliability validity. This study can be replicated. Ninka, the is freely available for anyone to use, including the source code files containing the sentence token expressions and token matching rules. Additionally, the scripts used to extract sentence token signatures and sentence token variability, and the spreadsheet output of each sentence token analyzed have been made available for replication at: http://turingmachine.org/2015/mit-bsd.

9 Conclusion

Licensors change the text of standard open source licenses for many purposes, including customizing the license with their specific organization's name, adding or removing conditions, and changing spelling or punctuation. Open source licensing standards like SPDX may be affected by the variability in licenses, as the variability may alter the legal meaning of licenses, creating legal issues in matching an altered license. In contrast, requiring an overly strict "perfect match" of open source licenses to the standard may result in the exclusion of many license texts with negligible variability. This paper presented an empirical study of the extent that the BSD and MIT/X11 family of licenses vary from their original definition. The BSD family of licenses closely match the existing SPDX templates, with little additional variability. The MIT/X11 family of licenses was found to be much more fragmented and heavily customized, including the creation of several specialized variants based from the original X11 license, customization of the text "authors or copyright holders", spelling alterations, and the adding and removing of conditions, grants and whole sentences. Small changes to the SPDX template for the MIT license, and to the SPDX list of equivalent words [7] would accommodate some the essential variation found within the license at a low cost.

References

1. German, D., Penta, M.D.: A method for open source license compliance of java applications. IEEE Software **29**(3), 58–63 (2012)
2. German, D.M., Manabe, Y., Inoue, K.: A sentence-matching method for automatic license identification of source code files. In: 25nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2010) (2010)
3. Gobeille, R.: The FOSSology project. In: MSR 2008: Proceedings of the 2008 International Conference on Mining Software Repositories, pp. 47–50. ACM, New York (2008)
4. Gonzalez-Barahona, J.M., Robles, G., Michlmayr, M., Amor, J.J., German, D.M.: Macro-level software evolution: a case study of a large software compilation. Empirical Software Engineering **14**(3), 262–285 (2009)
5. Lovejoy, J., Odence, P., Lamons, S.: Advancing the Software Package Data Exchange: An update. International Free and Open Source Software Law Review **2**(2), 145–152 (2013)
6. Rosen, L.: Open Source Licensing: Software Freedom and Intellectual Property Law. Prentice Hall (2004)
7. Stewart, K., Odence, P., Rockett, E.: Software Package Data Exchange (SPDX) Specification. International Free and Open Source Software Law Review **2**(2), 191–196 (2010)

The Right to a Contribution: An Exploratory Survey on How Organizations Address It

Germán Poo-Caamaño(✉) and Daniel M. German

University of Victoria, Victoria, Canada
{gpoo,dmg}@uvic.ca

Abstract. Free and Open Source Software (FOSS) projects are characterized by the opportunity to attract external contributors, where contributions can be in any form of copyrightable material, such as code or documentation. In most of them it is understood that contributions would be licensed in similar or compatible terms than the project's license. Some projects require a copyright transfer from the contributor to an organization for the work contributed to a project, such documents are known as copyright assignment agreements. In a way, it is similar to the copyright transfer than some researchers grant to a publisher. In this work we present an exploratory survey of the multiple visions of copyright assignments, and aggregate them in a work that researchers and practitioners could use to get informed of the alternatives available in the literature. We expect that our findings help inform practitioners on legal concerns when receiving external contributions.

1 Introduction

In FOSS projects, contributions can come from anybody. When a project receives a contribution, how can it guarantee that it has the rights to incorporate such contribution into the project?

In this paper, we present findings from our exploratory literature survey on Copyright Assignments and Contribution License Agreements in FOSS projects. The available literature is scattered, no comprehensive body of research has been forthcoming to help practitioners and researchers cope with the different perspectives on the topic of these type of agreements. Reports on proliferation of such agreements in FOSS projects [2,21] motivated us to better understand their differences, the rationale behind their decisions, and the relevant literature. We were also motivated to study the impact of such agreements in the software development practices, and possible consequences for FOSS projects.

The contributions of this paper are an integrated overview of the available material; first, by reviewing the relevant literature on copyright assignments; and, second, by suggesting areas of future research. We expect this document to help FOSS developers both those who control and those who contribute regarding the legal concerns of accepting and submitting contribution.

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 157–167, 2015.
DOI: 10.1007/978-3-319-17837-0_15

2 Methodology

We approached this exploratory study as a literature survey on the current research and practices. This study is based on the original guidelines for systematic review in software engineering by Kitchenham [18]. To find the current practices, we extended our search beyond the scope of journals and conferences. The steps in the systematic review method used are: formulate research questions, carry out a search process, define inclusion and exclusion criteria, and analyze data.

The research questions addressed by this study are:

Q_1. How is the term "Copyright Assignment" defined?
Q_2. What research topics are addressed in the Copyright Assignments literature?
Q_3. What projects and organizations require an agreement?

The search process was a manual search on conference proceedings, journal papers, and projects' websites. We searched peer reviewed papers, essays, copyright assignment and contribution license agreement forms, and FOSS projects' policies on copyright assignments. We searched for *'copyright assignment'* and *'contribution license agreement'* in combination with *'free software'* and *'open source software'*.

The selected literature body should commit to a set of inclusion criteria: *a*) the literature should address the area of contributions agreements, either main or secondary; *b*) be either an agreement form with legal implications (in any format) or have a document body in the form of research paper, essay, or book; and, *c*) be written in English.

We classified the agreements according the their clauses, and grouped them depending on whether they would require transferring the copyright of the contribution or not.

3 Results

In this section, we present the analysis of the literature and discuss the answers to our research questions.

3.1 How Is the Term "Copyright Assignment" Defined?

The term Copyright Assignment has been used in other contexts than FOSS, such as the publishing and music industry. Wu [34] defines Copyright Assignment as "*the transfer of the rights of the author to another party [...] For example, large record labels usually demand that composers assign all of their copyrights to the label*".

In the context of FOSS, we found that few authors define Copyright Assignment. In those cases, Copyright Assignment is defined together with Contribution License Agreement, both of them as a subclass of Contributor Agreement. We, therefore, present the definitions of Contributor Agreement, Copyright Assignment and Contributor License Agreement.

Guadamuz and Rens [14] define Contributor Agreements as "*written contracts in which the contributor either assigns copyright to the project organisers, or grant them a license*". Depending on how the document is drafted, the resulting contract would be either a Copyright Assignment Agreement or Contributor License Agreement. When the contributor assigns copyright—transfer the copyright ownership— to the project owner, it is a Copyright Assignment Agreement. When the contributor grants and irrevocable license—which may be exclusive or non–exclusive— to the project owner to use the contribution, then it is a Contributor License Agreement.

Jakob [15] describes a Contributor Agreement as a way to "*regulate the relationship of the developer with a particular organizational entity*". This agreement could either be a Copyright Assignment Agreement "*whereby the developer transfers and abandons his intellectual property rights in the contribution for the benefit of a project's administration*", or a Contributor License Agreement "*whereby the developer is only required to grant usage rights*".

Maracke [21] describe a Contributor Agreement as a mechanism to "*define and clarify the terms, under which a contribution (code, translation, artwork, etc.) is made to an open source or open content project*". Later, she defines altogether "*Assignment [A]greements require the assignment and therefore transfer of copyright in all contributions to the project owner, while [L]icense [A]greements grant an irrevocable license to allow the project owner to use the contribution*".

As seen above, Copyright Assignment is usually done via a Copyright Assignment Agreement (CAA). An alternative to the Assignment is a Copyright License Agreement (CLA) where the owner does not transfer the copyright and instead provides a license to the recipient of the contribution.

3.2 What Research Topics Are Addressed in the Copyright Assignments Literature?

During the analysis of the literature, we could identify five categories or topics covered by the surveyed literature. Table 1 shows a summary of the literature according their topic focus. Below we elaborate on the five categories.

Table 1. Papers according to topics covered

Category	Literature (research papers, essays, books)
Governance	[12,13,24,27,29,30,33]
Community Building	[1,2,13,20,24]
Litigation	[9,13,29,33]
Business Model	[6,13,17,23,29,33]
Proliferation of Agreements	[2,4,7,19,21,25]
Other	[3,5,8,10,14–16,26,28,31]

Governance. Markus [22] defines governance in FOSS as "*the means of achieving the direction, control, and coordination of wholly or partially autonomous individuals and organizations on behalf of an OSS development project to which*

they jointly contribute". These papers describe the impact that a CAA or CLA have on the governance of a project.

Wielsch's [33] addresses the challenges that projects face when there are several authors or copyright holders. The legal aspects of licenses and copyright assignment, and the role that some clauses might have in the governance of open collaborative works, not only FOSS projects.

Sometimes the license chosen for a project produces conflicts, such as incompatibilities with other FOSS projects that use other licenses. Such incompatibilities can be an obstacle to reuse code from other projects (either by linking to them, or by copying their source code) [11]. Either changing or adding a small exception to a license may require a substantial effort to contact and receive consent from all authors (the current copyright holders of the project). Such efforts can be avoided if the copyright is hold by a single entity that can decide the changes unilaterally [13]. Therefore, a single legal owner is a single point of contact and can give flexibility to a project [24], because this also yields power to produce changes to the licensing of the project whenever is needed [13]. The single owner becomes the steward of the project.

Many FOSS projects have informal structures, sometimes without a legal entity or sponsor behind them. The lack of a formal entity creates uncertainty on who has the authority on the project and who takes decisions, particularly, licensing decisions [29]. The legal entity can be a foundation or a company that acts as sponsor of the project. Sponsored projects can be perceived as mature, or at least more than community driven projects. Such maturity can be in the code base, organizational structure of the project, financial backing or a more established base of existing developers [32].

In sponsored projects, the ownership is associated with the degree of responsibility in the project [30], and sometimes, outside contributors can not participate in the design phase that defines the future of the project [32].

Contributors—other than the steward—might consider that this centralization creates uncertainty about the future of the project. The steward may not act predictably according to the interests of the contributors of a project, as organizations depend on the shareholders or management. Although uncommon, there is evidence of organization switching strategies. Caldera, a company known as a contributor of Linux, after being acquired by SCO, switched strategies and litigated copyright infringement in the Linux source code against IBM [24]. After the dissolution of X Consortium, a non-profit organization, the copyright was transferred to its successor organization, The Open Group. The new organization *"changed all the MIT/X Consortium rights to a restrictive copyright"* [12]. Similar concern were raised when Oracle acquired Sun Microsystems; the later had acquired MySQL AB, the original steward of the MySQL project [26].

In a project with disparate ownership each contributor has a valuable ownership right, with different kind of interests in the future of the project. In other words, external contributors, either individuals or companies, may react negatively to sponsored projects perceived as tightly controlled by the copyright holder and not truly participating in collaborative partnerships [24,32].

Changing a license might have a positive outcome, such as facilitate code reuse among different projects. However, the code may also be relicensed in a way that limit its reuse among certain FOSS projects, or even limit the project's ability to reuse FOSS projects. For example, even thought LibreOffice and OpenOffice are both derived from the same code base, each has different licenses; as a consequence LibreOffice can reuse code from OpenOffice, but not the other way around [10].

Community Building. Some organizations decide to release their source code to increase adoption of their product(s), or to encourage the creation of standards around a certain technology, eventually built by them [32]. These papers describe the impact that a CAA or CLA have on community building.

As a copyright holder can license its work any number of times, the steward can license the software under multiple licenses, either FOSS, proprietary (non-FOSS) or both. Having multiple FOSS licenses can help projects interested in a wide adoption of their code, that is, to be used by other FOSS projects with different licenses.

To build a community, FOSS projects sponsored by an organization can create a tension between the organization and external contributors, because of the sense of control and ownership [32].

For volunteers, it is motivational to have their contributions accepted and merged into the code base. This motivation might decrease if it takes time for the contributions to be incorporated. Although this could happen with or without copyright assignment [1], legal paperwork may undermine the motivation to contribute [13,24,32].

Copyright assignment introduces asymmetry in the relationship between the copyright holder and outside contributors. They are legally unequals [6]. For example, the copyright holder can create a stream of revenue by implementing a dual license for the project. Although a valid approach for business model, it might hurt the ability to nurture a community because the revenue stream is unique to the copyright holder. This inequality creates a barrier to involvement by other contributors, like companies [10,31].

Submitting a contributor agreement can take significant time because those agreements have to be reviewed, approved and signed; and when the contributor is an employee or a company, it is likely to be done by management and legal departments of such organization [2].

Litigation. These papers describe the impact that a CAA or CLA have on litigation.

Rosen [29] covered copyright assignments, dual license and litigation in FOSS projects in his work. He also explains joint work as a way to relicense software and the practice in some FOSS projects to assign ownership to a steward, and some of the risks that a developer could face by assigning the copyright, such as the original authors lose the right to relicense their own work or the perils of assigning the copyright to an informal entity.

License enforcement is often mentioned as the primary reason why organization like the FSF require developers to assign copyright. Under US law, only the owner of a copyright has the right to enforce the license [9,29]. Centralizing the copyright in a single owner simplifies enforcement for the entire code base and, at the same time, releases the contributors of this burden [13]. It also helps in registering copyrights in jurisdictions where it is required [33].

In some cases a copyright assignment is used to circumvent limitations of a FOSS license. For example, to negotiate indemnification when the license in use has patent termination clauses, like MPL1.1 (section 8.2), MPL2.0 (section 5.2), CPL (section 7), OSL/AFL (section 10) or GPLv3 (section 11), among others [29].

License enforcement could be also used to obtain economical advantage. If someone is infringing a GPL code, then the organization holding the copyright could negotiate the terms for a proprietary license. Yet, this might be rarely the desired outcome for developers who are not employees of the organization holding the copyright [13].

Business Model. Licensing the same product under multiple licenses (FOSS and proprietary) might be mechanism used by the licensor to sell proprietary licenses for a fee. These papers describe the impact that a Copyright Assignment has when it is used as a business model.

A licensor could offer two versions of the same product: a commercial one and a FOSS one. The former can be offered with additional features, support, more elaborated warranties or forms of indemnification, or any other negotiation. The latter can be used to promote the software or get external contributions. However, a licensor can only license software for which it holds the copyright or have received permission to license. Therefore, having a single entity holding the copyright of a product can be attractive as a business model [29].

When a single entity holds the copyright of the product, and the product is licensed with a copyleft license, then that entity holds an exclusive power that no one else in the community holds [13]. In this context, the project becomes an asset for the steward. Only the steward can sell and distribute the community's work under creating proprietary licenses (usually adding commercial support, or other services that allow them to monetize their investment in a FOSS project). This asset can be sold total or partially, used to negotiate acquisition, or disposed to the higher bidder in case of bankruptcy" [24]. However, the value might depend on effective ways of verifying the code does not contain "unexpected FOSS or other necessary contractual cover" [17].

However, the circumstance previously discussed does not occur in projects where the license is academic, because in those cases, everyone has equal right to license proprietary versions. This could explain why copyright assignment makes more sense when they are used in projects that use a copyleft licenses [23].

Proliferation of Agreements. There had been a proliferation of copyright and license agreements in FOSS projects. Although these agreements might have

similar goals they differ in the wording. These papers describe the problem and challenges of proliferation of agreements.

A consequence of having different agreements is that all of them have to be analyzed by lawyers who assist developers, projects, companies and non-profit organizations. Instead, Brock [2] proposed to unify all kind of copyright and license agreements into one, either by using a single modular document with extensive options or multiple documents. However, it raised some criticism because with this initiative the transfer of rights could be perceived as an industry standard [7,19,21].

Metzger [25] studied the duration and territoriality of copyright and license agreements. Engelhardt [4] compared their legal consequences in different jurisdictions, and how they differentiate the wording applied in these agreements. Thus, the enforcement might require adjustments in the agreements to match different local copyright laws. However, such effort can augment the proliferation of agreements, and challenge the initiatives that try to standardize them.

3.3 What Projects and Organizations Require Copyright Assignment?

Because of the proliferation of agreements [2], we searched for organizations that require that a contribution is accompanied by its corresponding copyright agreement (including assignments)[1] before such contribution is accepted and merged. We classified them according to their intent, either as CAA or CLA. We examined every agreement (we wanted to avoid cases in which a Copyright Assignment was labelled as a Contributor License Agreement, and the other way around). To enable further research, we included historical agreements that organization does not require anymore, such as OpenOffice, Evolution, Clutter and cogl. Table 2 comprises the agreements separated by type of organizations (non–profit and for–profit), as it has been reported to make a difference [19,24,30].

4 Discussion

It is dangerous for a FOSS project to assume that it has the rights to distribute an external contribution, in particular if such contribution is significant. As it has been described above, the organizations (and individuals) behind FOSS projects have taken two different approaches to address this issue: assigning copyright (copyright assignment agreements–CAA) or requesting a license to distribute the contribution (copyright licensing agreement–CLA). Each model has its advantages and disadvantages.

CAA centralize the ownership of the project. The steward (its owner) is capable of relicensing the code, either by changing its FOSS license, or relicensing under a proprietary license. This model seems to be preferred by organizations

[1] The agreements forms collected can be found at https://github.com/blindr/contributor-assignments.

Table 2. Organizations (and their respective projects) that require either Copyright Assignment Agreements (CAA) or Contribution. License Agreements (CLA).

Type	Projects (Organization)
Non–profit organizations	
CLA	Apache Software Foundation (*HTTPS Server, Tomcat, harmony, ...*), Diaspora (*Diaspora*), Django Software Foundation (*Django*), Eclipse Foundation (*Eclipse*), Mozilla Foundation (*Firefox, ...*), OuterCurve (*NuGet, ASP.NET Ajax Library, ...*), Perl Foundation (*Perl*), The PHP Group (*PHP-PDO*)
CAA	Free Software Foundation (*gcc, emacs, glibc, ...*), Open Source Matters (*Joomla*), The Mambo Foundation (*Mambo*)
For–profit organizations	
CLA	Canonical (*Unity, Bazaar, Launchpad, Upstart, ...*), Google (*Android, Chromium, GWT, V8, Go, ...*), Joyen (*Node.js*), Phrabicator (*Phacility*), Red Hat (*Fedora*), Zend Technologies (*Zend*)
CAA	ArtofCode (*Ghostscript, libart, ...*), Intel (*Clutter, cogl*), Nokia/Digia/Qt Company (*Qt, S60/Symbian, ...*), Novell/Xamarin (*Mono*), Openfiler (UK) Ltd (*Openfiler*), Oracle (*Java, MySQL, OpenOffice.org*), Red Hat (*Cygwin*), Rich Hickey (*Clojure*), VMware (*Zimbra, Open VM tools*), Ximian/Novell (*Evolution*)

that use copyleft licenses (such as the GPL). Examples of such organizations are: the Free Software Foundation, who requests copyright in order to be able to enforce its copyright (on behalf of its authors); for-profit companies such as Nokia, Sun Microsystems, Oracle and Red Hat request a CAA as part of their business model, which allows them to maintain full ownership of their software, and sell proprietary licenses to it.

On the other hand, projects that use permissive (academic) licenses are more likely to require CLAs. The argument can be made that, because anybody can use the software to create other proprietary and FOSS software, the copyright does not give them any strategic advantage. As shown in Table 2, most non-profit use CLAs rather than CAAs.

In both models, we can find variations in the wording and the way the agreement must be filled and/or signed. Depending of the jurisdiction, a "technical signature" (for example, click on a button) is sufficient for most cases. In others, a paper based signature is required [14]. Chooding a model might depend of the interests of the project, and the alternatives available in the lost jurisdiction.

It is possible that CAAs reduce the number of potential contributors, since those not willing to assign copyright would not participate. It is also known that forks have been created when the copyright of significant contributions was not assigned (as is the case with xemacs, a fork of emacs). One can argue that organizations that use CLAs instead of CAAs create a more egalitarian environment. Further research is required to fully understand the impact of the choice of agreement.

Some copyright assignments give back a license to the contribution which grants the author the same benefits as a copyright holder. This includes rights not granted by the project's FOSS licenses, such as permission for proprietary relicensing, except granting exclusive rights (since it was already transferred, it is not exclusive anymore [8]). This will allow the author to contribute the code to another project (under a CLA, but not under another CAA) [29].

Another aspect that is worth considering is, how big a contribution should be before one should worry about the right to use it? For instance, does it require permission to redistribute simple bug fixes? The Free Software Foundation does not require copyright assignment for contributions that involves less than 15 lines of code. If a contributor makes a series of repeated small changes, then it could become a significant contribution, for which it would be required to sign a CAA before accepting further contributions [9].

Another aspect corresponds to the legal consequences of copyright as intellectual property. As such, it is inheritable upon death of the owner, the owner might prefer to assign or transfer the rights to an organization that could manage them instead of risking an involuntary lost [29]. In addition, a contributor assignment might last different depending on the territory, according to the expiration of copyright in each legislation.

5 Limitations of This Study

This study is based on the guidelines for systematic review by Kitchenham [18], but it deviates from the guidelines in several ways: *a*) the search was manual, and *b*) the search scope was wider than journal and conferences papers, including books, essays, and legal documents. As a consequence, we may have missed some relevant studies or overlooked the literature. However, the consolidation of agreements and the classification of the literature, can help towards an empirical study of the impact of such agreement requirements in FOSS projects.

The authors are not lawyers and they might have misinterpretations of the consequences of the agreements analyzed. Whenever possible, we have provided references to legal expects supporting our assertions. However, this research would benefit from a review by lawyers expert in FOSS.

6 Conclusions and Future Work

We surveyed and aggregated multiple visions of copyright assignments. We also collected copyright assignments and contributor license agreements forms from multiple FOSS projects. We learned that accepting contributions pose legal challenges to developers, companies and non-profit organizations involved in FOSS projects. There are several alternatives that require an agreement before accepting contributions, each has its benefits and drawbacks. The expectations of the steward of the project and external contributors might differ, and the right solution will depend on the goals of each project and the specifics of each contribution.

Future work should aim at investigating projects that have stopped requiring the copyright assignment for contributions, in particular, the reasons behind such decision. Also important is to study the rationale of projects that were forked because of the requirement of a copyright assignment. Another aspect that requires further study is whether some individuals might decide not to contribute due to such agreements, and, similarly, what is the impact of such agreements in the dynamics between contributors and the steward of a project. Overall, we expect that our findings help inform practitioners about legal concerns when receiving external contributions, and enable researchers and practitioners to get informed of the alternatives available in the literature.

References

1. Allyn, M.R., Misra, R.B.: Motivation of Open Source Developers. Intl. Journal of Open Source Software and Processes **1**(4), 65–81 (2009)
2. Brock, A.: Project Harmony: Inbound transfer of rights in FOSS Projects. Intl. Free and Open Source Software Law. Review **2**(2), 139–150 (2010)
3. Brock, A.: Understanding Commercial Agreements with Open Source Companies. In: Thoughts on Open Innovation: Essays on Open Innovation from Leading Thinkers in the Field, pp. 119–139. OpenForum Europe LTD for OpenForum Academy (2013)
4. Engelhardt, T.: Drafting Options for Contributor Agreements for FOSS: Assignment, (Non)Exclusive Licence and Legal Consequences. A Comparative Analysis of German and US Law. SCRIPTed **10**(2), 149–176 (2013)
5. Fogel, K.: Producing Open Source Software: How to Run a Successful Free Software Project. O'Reilly Media, Inc. (2005)
6. Fontana, R.: Contributor Agreements Considered Harmful. Audio of talk given at OSCON 2011. Online (July 2011). http://faif.us/cast/2011/aug/30/0x17/
7. Fontana, R.: The Trouble With Harmony. Online (July 2011). http://opensource.com/law/11/7/trouble-harmony-part-1
8. Fontana, R., Kuhn, B.M., Moglen, E., Norwood, M., Ravicher, D.B., Sandler, K., Vasile, J., Williamson, A.: A Legal Issues Primer for Open Source and Free Software Projects. Software Freedom Law Center (2008)
9. Free Software Foundation: Information for maintainers of GNU software. Online (November 2012). http://www.gnu.org/prep/maintain/ (accessed: December 8, 2012)
10. Gamalielsson, J., Lundell, B.: Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved? Journal of Systems and Software **89**, 128–145 (2014)
11. German, D.M., Hassan, A.E.: License integration patterns: Addressing license mismatches in component-based development. In: 31st Int. Conf. on Soft. Eng., ICSE, pp. 188–198 (2009)
12. Jim, G.: Copyright Assignments. Online (July 2000). https://mail.gnome.org/archives/foundation-list/2000-July/msg00332.html (accessed: December 9, 2012)
13. GNOME: GNOME Foundation Guidelines on Copyright Assignment. Online (July 2010). https://live.gnome.org/CopyrightAssignment/Guidelines (accessed: December 9, 2012)

14. Guadamuz, A., Rens, A.: Comparative Analysis of copyright assignment and licence formalities for Open Source Contributor Agreements. SCRIPTed **10**(2), 207–230 (2013)
15. Jakob, S.F.: A Qualitative Study on the Adoption of Copyright Assignment Agreements (CAA) and Copyright License Agreements (CLA) within Selected FOSS Projects. JIPITEC **5**(2), 105–115 (2014)
16. Jensen, C., Scacchi, W.: License update and migration processes in open source software projects. In: Hissam, S.A., Russo, B., de Mendonça Neto, M.G., Kon, F. (eds.) OSS 2011. IFIP AICT, vol. 365, pp. 177–195. Springer, Heidelberg (2011)
17. Kemp, R.: Current developments in OSS. Computer Law & Security Review **25**(6), 569–582 (2009)
18. Kitchenham, B.: Procedures for performing systematic reviews. Tech. rep., Keele University (TR/SE-0401) and National ICT Australia Ltd. (0400011T.1) (2004)
19. Kuhn, B.: Project Harmony Considered Harmful. Online (July 2011). http://ebb.org/bkuhn/blog/2011/07/07/harmony-harmful.html
20. Lerner, J., Tirole, J.: Some Simple Economics of Open Source. The Journal of Industrial Economics **50**(2), 197–234 (2002)
21. Maracke, C.: Copyright Management for Open Collaborative Projects-Inbound Licensing Models for Open Innovation. SCRIPTed **10**(2), 140–148 (2013)
22. Markus, M.L.: The governance of free/open source software projects: monolithic, multidimensional, or configurational? Journal of Management & Governance **11**(2), 151–163 (2007)
23. Mcgowan, D.: Legal aspects of FOSS. In: Perspectives on FOSS, ch. 19, pp. 361–391. MIT Press (2007)
24. Meeks, M.: Some thoughts on copyright assignment. Online (December 2009). http://people.gnome.org/~michael/blog/copyright-assignment.html (accessed: April 27, 2011)
25. Metzger, A.: Internationalisation of FOSS Contributory Copyright Assignments and Licenses: Jurisdiction-Specific or "Unported"? SCRIPTed **10**(2), 177–206 (2013)
26. Moglen, E.: Software Freedom Law Center Opinion on the Oracle/Sun Merger (2009). https://softwarefreedom.org/news/2009/dec/04/software-freedom-law-center-submits-opinion-oracle/
27. O'Mahony, S.: Guarding the commons: how community managed software projects protect their work. Research Policy **32**(7), 1179–1198 (2003)
28. O'Mahony, S.: Nonprofit Foundations and Their Role in Community-Firm Software Collaboration. In: Perspectives on FOSS, ch. 20, pp. 393–413. MIT Press (2005)
29. Rosen, L.: Open Source Licensing: Software Freedom and Intellectual Property Law. Prentice Hall (2004)
30. Shuttleworth, M.: The responsibilities of ownership. Online (July 2011). http://www.markshuttleworth.com/archives/687 (accessed: December 9, 2012)
31. Smith, D., Alshaikh, A., Bojan, R., Kak, A., Kohe, J.M.: Overcoming Barriers to Collaboration in an Open Source Ecosystem. Technology Innovation Management Review **4**, 18–27 (2014)
32. West, J., O'Mahony, S.: Contrasting Community Building in Sponsored and Community Founded OSS. In: Proc. of the 38th Intl. Conf. on System Sciences, p. 196c. IEEE (2005)
33. Wielsch, D.: Governance of Massive Multiauthor Collaboration. Journal of Intellectual Property, Information Technology and E-Commerce Law **1**(2), 96–108 (2010)
34. Wu, T.: On Copyright's Authorship Policy. The University of Chicago Legal Forum **2008**, 335–354 (2008)

OSS 2015 Ph.D. Contest

Open Source Software Ecosystems: Towards a Modelling Framework

Oscar Franco-Bedoya[1,2(✉)]

[1] Group of Software and Service Engineering (GESSI), Universitat Politècnica de Catalunya, Barcelona, Spain
ohernan@essi.upc.edu
http://www.essi.upc.edu/∼gessi/
[2] Universidad de Caldas, Manizales, Colombia

Abstract. Open source software ecosystem modelling has emerged as an important research area in software engineering. Several models have been proposed to identify and analyse the complex relationships in OSS-ecosystems. However, there is a lack of formal models, methodologies, tool support, and standard notations for OSS-ecosystems. In this paper we propose a general framework for support the OSS-ecosystems modelling process. This framework will allow the representation, synthesis, analysis, evaluation, and evolution of OSS-ecosystems. Design science methodology is proposed to create several artefacts and investigating the suitability of these artefacts in the OSS-ecosystem context.

Keywords: Modeling · Software ecosystem · Framework · Open source software

1 Introduction

Software ecosystem modelling (SEM) is one of the most studied areas in the software ecosystem domain [1]. However, currently there is no established modelling standard for software ecosystems. Nonetheless, there exists a need for modelling software ecosystems because:

- *Complexity.* Software ecosystems are among the most complex systems ever created by human [2], and models may help understanding them.
- *Traceability.* The software industry is constantly evolving and is currently undergoing rapid changes [3]. Models help in tracing the historic software ecosystem changes.
- *Communication.* Models help to represent the complex network of symbiotic relationships between entire social actors, open source communities and commercial software companies, etc. [4].

Ecosystem stakeholders need a common language to communicate their ideas. In the particular case of OSS Software Ecosystems, to our knowledge, there

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 171–179, 2015.
DOI: 10.1007/978-3-319-17837-0_16

are no contributions in the literature regarding how to model OSS-ecosystem support the representation and analysis of the specific relationships between OSS-ecosystem actors. The aim of the present work is to make a characterization of OSS-ecosystems, and to propose a specific framework for modelling OSS-ecosystems. This framework will be able to: represent, evaluate, evolve and analyse OSS-ecosystems.

2 Related Work

2.1 Software Ecosystems

The field of biological ecosystems has inspired several ecosystem domains. The business ecosystem (BECO) term was coined by Moore in 1996 [5]. The term digital business ecosystems (DBECO) was obtained by adding digital in front of business ecosystem [6]. the matured concept of DBECO was exposed by Briscoe that defined DBECO as distributed adaptive open socio-technical systems, with properties of self-organization, scalability and sustainability, inspired by natural ecosystems [7]. The term software ecosystem (SECO) has been coined by Messerschmitt and Szyperski to refer to *"a collection of software products that have some given degree of symbiotic relationships"* [8], [2]. However, in contrast to natural ecosystem, there is no common definition of software ecosystem. It can be defined and interpreted in different ways, depending on the point of view [2]. Some of the most accepted definitions of SECO are:

- A software ecosystem is *"a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them"* [1].
- A software ecosystem is *"a collection of software projects which are developed and evolve together in the same environment"* [9].
- A software ecosystem is *"a set of software solutions that enable, support and automate the activities and transactions by the actors in the associated social or business ecosystem and the organizations that provide these solutions"* [10].

2.2 Open Source Software Ecosystems

According to our knowledge of the literature, there are only a few definitions of OSS-ecosystem. These are provided by Wynn [11] and Hoving et al. [12]. Both authors define OSS-ecosystem based on the Iansiti and levien concept of business ecosystem (BECO) [13]. This mean that the shared market, organizations, capital, are the main actors that support the open source software community. In addition Jansen highlights the role of the developers in the OSS community and the freely nature of resources in its definition [12]. These concepts are a weak metaphor of natural ecosystem terms. However, the transfer of knowledge has essentially limited itself to a reuse of terms [2]. In order to support our proposal, In Table 1 we have integrated the different concepts of ecosystems:

Table 1. Open Software Ecosystem Definition

Definition	Sources	Examples
An OSS-ecosystem is a SECO placed in a heterogeneous environment	Iansity and Levin BECO	Other OSS ecosystems, commercial SECOs,Government, Market rules, etc.
Its boundary is a set of niche players	Jansen SECO	Partners, Re-sellers, Platform provider, etc.
The keystone player is an OSS community around a set of projects in a common platform	Lungu SECO	Contributors, passive users, data sources, etc.

OSS-ecosystems are complex artefacts that require a specific characterization in order to model its elements and relationships. The variety of ecosystem described above have common roles, (e.g. partners, users, developers, resellers, software products and services, etc.). However, in the OSS-ecosystems there are several conceptual and structural particularities, e.g., principles, community join process, goals, governance, legalities, among other concepts. A characterization conducted to evaluate similarity between commercial SECOs and OSS-ecosystems is described in detail in sections below.

2.3 Software Ecosystem Models

In the literature several specific models and meta-models have been proposed to identify and analyse the relationships between software ecosystems members. The work of Yu and Deng [3] and Lopez et al. [14] use i* models to model the strategic dependencies between OSS-ecosystem actors. Boucharas et al., [15] and Jansen et al., [1] present the software ecosystem modelling (SEM) technique, which includes the product deployment context (PDC) and software supply networks (SSN) diagrams. A framework for sustainable software ecosystem management was discussed by Dhungana et al, [16]. Other authors like [17] and [18] represents the OSS-ecosystem actors and relationships using conceptual maps.

3 Research Methodology

We structure our research in terms of design science since it involves creating new artefacts and acquiring new knowledge, using an engineering cycle as main cycle and internal iterations with engineering activities and the empirical cycle Wieringa [19]. In our project the engineering cycle and the empirical cycle consist of five phases:

– *Problem investigation.* To investigate the nature of the problem we need to solve and to know which actions can help solve this problem.

- *Treatment design.* To design the solution for the identified problem, in this phase it is necessary to evaluate several alternatives for each designed solution.
- *Design validation.* Before we construct the framework's components, we validate the designs to assure that the selected designs satisfy the criteria for the framework components.
- *Treatment implementation.* To realize the solution specification for the problem we will develop the framework, some components will be conceptual components and others software components.
- *Implementation evaluation.* Wieringa defines evaluation as the use of artefacts in context. We will validate the framework use case studies validation.

4 OSS-Ecosystem Modelling Framework

The ecosystem terminology defined by Mens [2] and Lopez [14] show that there are several differences between commercial SECOs and OSS-ecosystems. *Sustainability* in an OSS-ecosystem is related to the number of ecosystem community members [20]. On the other hand, in a commercial SECO it depends mainly on economic factors. The *adoption strategies* and the *adoption risks* derived by using OSS products in a company affect organizations business models [14]. In general, the most of the risks for adopting OSS components in an OSS-ecosystem are related to the licenses heterogeneity. *Governance* commercial software ecosystems are typically governed by a decision maker that decides how the ecosystem should evolve, while OSS-ecosystems often have a much more exible decisional structure [2]. The community is the *organizational* unit in OSS-ecosystems. In contrast, hierarchy structures are common in commercial SECO. An OSS-ecosystem modelling framework has to support: visualization, synthesis, analysis, evaluation and evolution of OSS-ecosystem models. In this section we provide a brief overview of the tools that support these activities. Our goal with this framework is to offer suggestions and ideas to researchers and practitioners in the field of OSS-ecosystem modelling. The framework that we propose is shown in Figure 1.

4.1 OSS-Ecosystem Model Synthesis

The purpose of this activity is to answer the question: *How is it possible to generate a specific OSS-ecosystem model only from OSS-ecosystem data sources?*. Figure 2 shows a layered view of the components for this activity. At the bottom, there are several types of OSS-ecosystem data sources. Jansen defines three types [21]: project web sites, ecosystem hubs and aggregation sites. We added two other kinds of data sources: (1) social media sites such as Twitter, Facebook, etc. (2) strategic data from people related to the OSS-ecosystem obtain using specialized surveys. The OSS-ecosystem communities, typically provide open access to all data sources. The extraction of data is done with dedicated tools developed by the OSS-ecosystem researchers. Occasionally it is done by the use

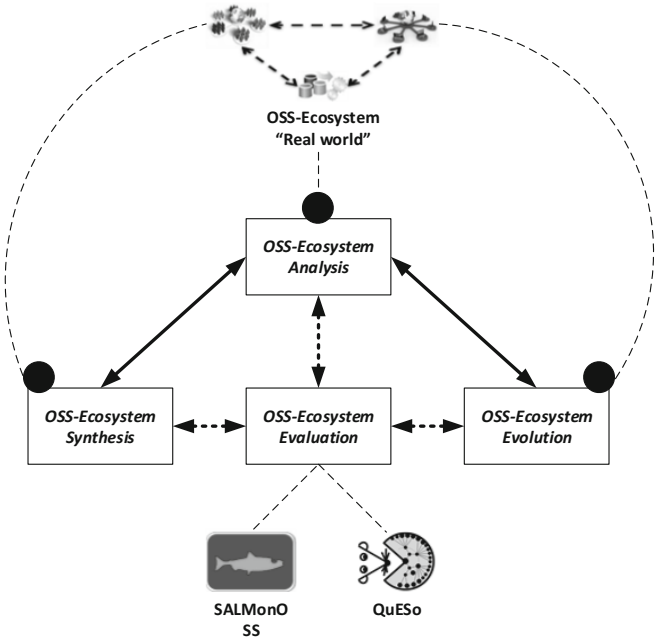

Fig. 1. OSS-ecosystem modelling framework

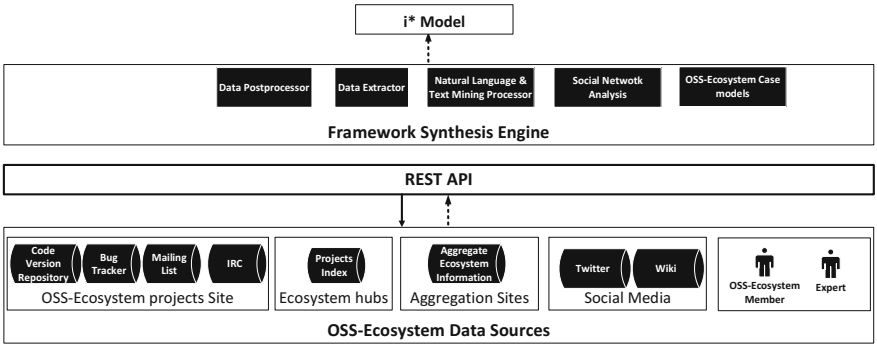

Fig. 2. OSS-Ecosystem Model Synthesis

of specialized tools. E.g. Spanish LibreSoft group provides FLOSSMetrics to extract data from repositories and then they are stored in a set of databases [22]. However, most of these tools are not reusable in other experiments, even in the same OSS-ecosystem. The availability of the data depends of the OSS-ecosystem. Because of this, we propose to define an extensible REST API. This is a set of web

services to be implemented by each OSS-ecosystem community. This interface would allow obtain information related to the OSS-ecosystem.

Our aim is to motivate the development of this API by the OSS-communities providing them with a framework for modelling, analysing and monitoring their own OSS-ecosystem. The framework synthesis engine has two extraction components, similar to Goemmine et al. [22]. Moreover, It uses the OSS ontology defined by Lòpez et al. [14], social network analysis (SNA), self-modelling techniques and predefined OSS-ecosystem models to identify the OSS-ecosystem actors and relationships in a specific OSS-ecosystem (e.g. Eclipse, Gnome, etc. . Finally, the synthesis components will generate an i* OSS-ecosystem model.

4.2 OSS-Ecosystem Model Valuation

This component will enable the monitoring of the OSS-ecosystem health. To prove the feasibility of the approach we propose develop this component based on an existing technologies named SALMonOSS [23] and QuESo [20] developed in our research group.

- *SALMonOSS.* The general idea is to adopt principles and methods from the service oriented computing field (SOC). Particularly, we propose to adapt the concepts of quality service and service level agreement, and propose to reuse the existing body of knowledge and techniques from SOC monitoring.Figure 3 shows the OSS-ecosystem evaluator. SALMonOSS is an OSS-ecosystem health monitor component able to: (1) monitor a list of ecosystem health indicators along time (2) link the gathered values with client's needs by operationalization of conditions in software ecosystem agreements (SELAs) and (3) engineer a portfolio of methods and techniques that supports OSS ecosystems (e.g. OSS selection, proactive adaptation, etc.).
- *QuESo.* QuESo is a quality model for assessing the quality of OSS ecosystems. QuESo have been organized in three dimensions: (1 those that relate to the platform around which the ecosystem is built, (2) those that relate to the community of the OSS-ecosystem and (3) those that are related to the ecosystem as a network of interrelated elements, such as projects or companies. We are using QuESo to define the key health indicators (KHIs) to be monitored by SALMonOSS. the SELAs and the software ecosystem level fulfilment (SELF are composed by KHIs).

4.3 OSS-Ecosystem Model Analysis

If there is a defined OSS-ecosystem model, what type of data or functionality should be changed in the OSS-ecosystem to satisfy the proposal model? Figure 4 shows the main components for this activity. To answer this question we will use: an OSS-ecosystem ontology, the expert system engine and the case base reasoning (CBR). The rules defined in the ontology allow reasoning about the class instances and their relationships obtained from the OSS-ecosystem data sources.

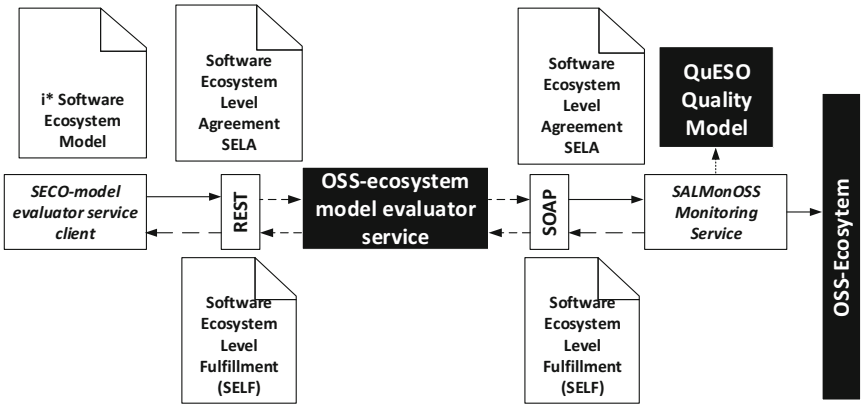

Fig. 3. OSS-Ecosystem Model Evaluation

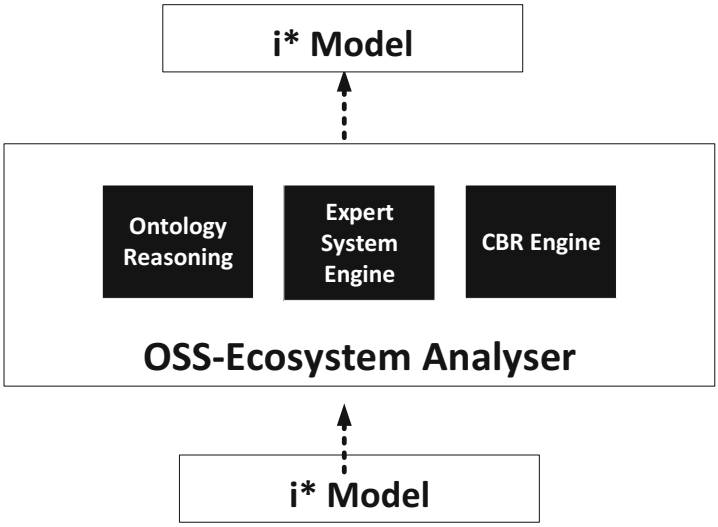

Fig. 4. OSS-Ecosystem Model Analysis

The expert system engine will be used to register the knowledge obtained from the software ecosystem experts about specific OSS-ecosystem models. Finally, we will use CBR reasoning to select strategies for propose possible changes in the initial OSS-ecosystem model defined.

4.4 OSS-Ecosystem Model Evolution

The OSS-ecosystem are dynamics and complex artefacts. Similar to the natural ecosystem the actors, roles, dependencies, resources, relationships, etc., changed

frequently over time. The question is: Once a specific OSS-ecosystem model has been created, how its continuous evolution along time can be done?. The software ecosystem evolution is stored in the OSS-ecosystem repositories. Since the software environment involves human beings (developers and users). This makes it possible, in principle, to interact with them in order to find out how and why a software project has evolved over time, and making it easier to alter the way in which the ecosystem will evolve in the future.[2]. With the tools defined in our framework and with the information stored in the OSS-ecosystem repositories, We will be able to visualize the changes in the OSS-ecosystem model structure, interactions, health, releases, resources, etc., all this from a social technical perspective. E.g. OSS-communities, legalities, partners, platform, technologies and projects.

5 Conclusions and Future Work

In this paper we have presented a general framework for representation, synthesis, analysis, evaluation and evolution of OSS-ecosystems. We believe that ecosystem modelling is a promising research direction and we plan to continue working on it. Our focus is would be defining methodologies, languages, formal syntax and semantic rules for modelling software ecosystem based on the models and metamodels described in the literature. In a first stage, we are working in the QuESo quality model validation and its integration with SALMonOSS framework.

Acknowledgments. This work is a result of the RISCOSS project, funded by the EC 7th Framework Programme FP7/2007-2013 under the agreement number 318249. We would also like to thank the contribution of EOSSAC project, founded by the Ministry of Economy and Competitiveness of the Spanish government (TIN2013-44641-P).

References

1. Jansen, S., Brinkkemper, S., Finkelstein, A.: Business network management as a survival strategy: A tale of two software ecosystems. In: Proceedings of the 1st Workshop on Software Ecosystems, CEUR-WS, pp. 34–48 (2009)
2. Mens, T., Claes, M., Grosjean, P., Serebrenik, A.: Studying evolving software ecosystems based on ecological models, pp. 297–326. Springer, Heidelberg (2014)
3. Yu, E., Deng, S.: Understanding software ecosystems: A strategic modeling approach. In: Proceedings of the 3th Workshop on Software Ecosystems, IWSECO, pp. 65–76 (2011)
4. Yamakami, T.: A three-layer view model of oss: Toward understanding of diversity of oss. In: Proceedings of the 13th International Conference on Advanced Communication Technology, ICACT, pp. 1190–1194 (2011)
5. Moore, J.F.: Predators and prey: a new ecology of competition. Harvard Business Review **71**, 75–83 (1993)
6. Stanley, J., Briscoe, G.: The ABC of digital business ecosystems. Computer, Media and Telecommunications Law **15**, 1–24 (2010) P28

7. Briscoe, G.: Digital Ecosystems. PhD thesis, Imperial College London (2009)
8. Messerschmitt, D.G., Szyperski, C.: Software Ecosystem: Understanding an Indispensable Technology and Industry. MIT Press Books, vol. 1. The MIT Press (2005)
9. Lungu, M., Malnati, J., Lanza, M.: Visualizing gnome with the small project observatory. In: Proceedings of the 6th IEEE International Working Conference Mining Software Repositories, MSR, pp. 103–106 (2009)
10. Bosch, J.: From software product lines to software ecosystems. In: Proceedings of the 13th International Software Product Line Conference, SPLC 2009, pp. 111–119. Carnegie Mellon University, Pittsburgh (2009)
11. Wynn Jr, D., Boudreau, M.C., Watson, R.: Assessing the Health of an Open Source Ecosystem. GI Publishing, New York (2008)
12. Hoving, R., Slot, G., Jansen, S.: Python: Characteristics identification of a free open source software ecosystem. In: Proceedings of the 7th IEEE International Conference on Digital Ecosystems and Technologies, DEST, pp. 13–18 (2013)
13. Iansiti, M., Levien, R.: The keystone advantage: what the new dynamics of business ecosystems mean for strategy, innovation, and sustainability. Harvard Business Press (2004)
14. López, L., Costal, D., Ayala, C.P., Franch, X., Glott, R., Haaland, K.: Modelling and applying OSS adoption strategies. In: Yu, E., Dobbie, G., Jarke, M., Purao, S. (eds.) ER 2014. LNCS, vol. 8824, pp. 349–362. Springer, Heidelberg (2014)
15. Boucharas, V., Jansen, S., Brinkkemper, S.: Formalizing software ecosystem modeling. In: Proceedings of the 1st International Workshop on Open Component Ecosystems, IWOCE, pp. 41–50. ACM, New York (2009)
16. Dhungana, D., Groher, I., Schludermann, E., Biffl, S.: Software ecosystems vs. natural ecosystems: learning from the ingenious mind of nature. In: Proceedings of the 4th ECSA, pp. 96–102. ACM (2010) P48
17. Morgan, L., Feller, J., Finnegan, P.: Exploring value networks. Eur J Inf Syst **22**, 569–588 (2013)
18. Mattmann, C.A., Downs, R.R., Ramirez, P.M., Goodale, C., Hart, A.F.: Developing an open source strategy for NASA earth science data systems. In: Proceedings of the 13th IRI, pp. 687–693. IEEE (2012) P15
19. Wieringa, R.: Design science as nested problem solving. In: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, pp. 8:1–8:12. ACM, New York (2009)
20. Franco-Bedoya, O., Ameller, D., Costal, D., Franch, X.: Queso: A quality model for open source software ecosystems. In: Proceedings of the 9th International Conference on Software Engineering and Applications, ICSOFT-EA, pp. 209–221 (2014)
21. Jansen, S.: Measuring the health of open source software ecosystems: Beyond the scope of project health. Information and Software Technology **56**, 1508–1519 (2014) Special issue on Software Ecosystems P68
22. Goeminne, M., Mens, T.: Analyzing ecosystems for open source software developer communities. In: Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry, pp. 247–275. Edward Elgar Publishing (2013) P57
23. Oriol, M., Franco-Bedoya, O., Franch, X., Marco, J.: Assessing open source communities' health using service oriented computing concepts. In: Proceedings of the 8th International Conference on Research Challenges in Information Science (RCIS), pp. 1–6. IEEE (2014)

Author Index