Open Source Development, Adoption and Innovation

Edited by
Joseph Feller
Brian Fitzgerald
Walt Scacchi
Alberto Sillitti

Springer

ifip

OPEN SOURCE DEVELOPMENT, ADOPTION AND INNOVATION

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 less 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 in information 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.

OPEN SOURCE DEVELOPMENT, ADOPTION AND INNOVATION

IFIP Working Group 2.13 on Open Source Software, June 11-14, 2007, Limerick, Ireland.

Edited by

Joseph Feller
Business Information Systems, University College Cork, Ireland

Brian Fitzgerald
Lero - the Irish Software Engineering Research Centre, University of Limerick, Ireland

Walt Scacchi
Institute for Software Research, Donald Bren School of Information and Computer Sciences, University of California, Irvine, USA.

Alberto Sillitti
Free University of Bolzano-Bozen, Italy

Springer

Library of Congress Control Number: 2007927182

Open Source Development, Adoption and Innovation

Edited by J. Feller, B. Fitzgerald, W. Scacchi, and A. Sillitti

p. cm. (IFIP International Federation for Information Processing, a Springer Series in Computer Science)

ISSN: 1571-5736 / 1861-2288 (Internet)
ISBN: 13: 978-0-387-72485-0
eISBN: 13: 978-0-387-72486-7
Printed on acid-free paper

9 8 7 6 5 4 3 2 1
springer.com

Preface

We are very pleased to introduce *Open Source Development, Adoption and Innovation,* the proceedings of the *Third International Conference on Open Source Systems* (OSS 2007). Open source software (free software, libre software) has emerged as a major field of scientific inquiry across a number of disciplines (software engineering, information systems, business, economics, law, sociology, just to name a few). In addition to this highly successful conference series, open source has been the focus of dozens of conference program tracks, workshops, tutorials and panels, as well as the theme for numerous special issues published by many of the top journals in the disciplines listed above. A substantial library of books has been published, targeted at both researchers and practitioners, and the web is awash with the experiences and perspectives of a large community of open source developers, observers, advocates (and detractors).

When the concept of open source began to gain mindshare in the global business community, decision makers faced a challenge: to convert hype and potential into sustainable profit and viable business models. As a community of researchers, we face a similar challenge: to convert anecdote and speculation into empirical description and predictive theory. Over the past several years, a substantial body of multi-disciplinary (and in some cases truly inter-disciplinary) literature has emerged. This literature provides us with rich descriptions of the various aspects of open source, many based on empirical data rigorously gathered and analyzed using both qualitative and quantitative methods. Our challenge is to continue this work, setting aside anecdote and insisting on rigor, depth and completeness in our research. We must also extend the work, so that open source research can move beyond description to understanding and prediction. Our rhetoric is filled with claims that open source is a different way to build software, acquire software, use software; that it challenges development processes, software business models, and intellectual property frameworks. Such rhetoric calls out for theory, both the careful testing of existing theory in new (open source) contexts and also the building of new theory emerging from the phenomenon itself.

We believe that this conference series, and the IFIP working group it represents, can play an important role in meeting these challenges, and hope that this book will become a valuable contribution to the open source body of research. Nearly 100 submissions from authors in 23 countries were carefully peer-reviewed; in the end, 15 papers are published here as full papers (Part I) and 25 published in a condensed form as short papers (Part II). Additionally, 12 paper authors presented their work in poster form for discussion at the conference (Part VI). Complementing the paper program, we are pleased to include descriptions of the two panels (Part III), three tutorials (Part IV) and five workshops (Part V) which were all an important part of the conference event.

About IFIP WG 2.13

This is the first conference since the formal acceptance of this new Working Group 2.13 on Open Source Software by IFIP. Since the formal founding meeting on 10 June 2006 in Como, Italy, we have sought to consolidate the working group. The Working Group has 27 founding members from Europe, North America, Middle East, Asia /Pacific and Africa. The main objective of the Working Group is to enable the diverse community of free, libre and open source software (OSS) researchers and practitioners to rigorously investigate the technology, work practices, development processes, community dynamics within OSS systems, complementing appropriately other IFIP Working Groups where OSS is increasingly relevant. The scope of the Working Group is detailed on the website which is being maintained by Kevin Crowston at http://www.ifipwg213.org.

Since the group has been founded, the have been several exciting developments in which group members have been centrally involved. Firstly, the results of several EU-funded initiatives are emerging, such as *CALIBRE* (www.calibre.ie), which studied the impact of open source software on the European secondary software sector, *COSPA* (www.cospa-project.org), which studied the adoption of open source by public administrations across Europe, and *FLOSSIMPACT* (www.flossimpact.eu) which studied the impact of free/libre/open source software on innovation and competitiveness of the European Union.

Ongoing projects in which the Working Group's members are central include *COSI* (www.itea-cosi.org), which is focusing on transferring the lessons of open source and agile methods to traditional in-house development (the so-called inner source phenomenon); *OPAALS* (www.opaals.org), which is seeking to build a sustainable interdisciplinary research community in the emerging area of digital ecosystems; QualiPSo (www.qualipso.org), which studies the quality and trust aspects of Open Source Software and its adoption in the main stream industrial process; and *Open Code, Content and Commerce (O3C) Business models*, a three-year investigation of open innovation and value-creation strategies.

IFIP WG 2.13 OFFICERS

General Chair:	Brian Fitzgerald, Lero, University of Limerick, Ireland
Vice-Chair:	Walt Scacchi, University of California - Irvine, US
Vice-Chair:	Giancarlo Succi, Free University of Bolzano-Bozen, Italy
Secretary:	Ernesto Damiani, Free University of Bolzano-Bozen, Italy

Acknowledgements

We gratefully acknowledge the contributions made by the other OSS 2007 Conference officials: Ernesto Damiani (Tutorial Chair), Scott A. Hissam (Workshop Chair), Sandra Slaughter (Panel Chair), Barbara Russo (Publicity Chair), and Alberto Colombo and Fulvio Frati (Webmasters). Special thanks go to Lorraine Morgan for her instrumental role in preparing this volume. We would like to thank the OSS 2007 sponsors, our international program committee, our board of reviewers and all the local volunteer organizers and supporters, without whom this conference could not take place. Finally, we thank all the authors, panelists, tutorial and workshop organizers who provided this years conference with such an excellent program.

Joseph Feller (Program Co-Chair)[1]
Brian Fitzgerald (Organizing Chair)[2]
Walt Scacchi (General Chair)[3]
Alberto Sillitti (Program Co-Chair)[4]

Limerick, Ireland
June 2007

[1] Business Information Systems, University College Cork, Ireland
[2] Lero - the Irish Software Engineering Research Centre University of Limerick, Ireland
[3] Institute for Software Research, Donald Bren School of Information and Computer Sciences, University of California, Irvine, USA.
[4] Free University of Bolzano-Bozen, Italy

Organization

Conference Officials

General Chair:	Walt Scacchi	University of California, Irvine, US
Program Chair:	Joseph Feller	University College Cork, Ireland
	Alberto Sillitti	Libera Università di Bolzano, Italy
Organising Chair:	Brian Fitzgerald	University of Limerick, Ireland
Tutorial Chair:	ErnestoDamiani	University of Milan, Italy
Workshop Chair:	Scott Hissam	Carnegie Mellon University, US
Panel Chair:	Sandra Slaughter	Carnegie Mellon University, US
Publicity Chair:	Barbara Russo	Free University of Bolzano-Bozen, Italy
Web Master:	Alberto Colombo	University of Milan, Italy
	Fulvio Frati	University of Milan, Italy

Program Committee

Guenter Boeckle	Siemens, Germany
Gerry Coleman	Dundalk Institute of Technology, Ireland
Kieran Conboy	NUI Galway, Ireland
Kevin Crowston	Syracuse University, USA
Jean-Michel Dalle	Universite Pierre et Marie Curie, France
Stefano De Panfilis	Engineering Ingegneria Informatica, Italy
Francesco Di Cerbo	University of Genoa, Italy
Gabriella Dodero	University of Bolzano-Bozen, Italy
Mahmoud Elish	King Fahd University, Saudi Arabia
Pat Finnegan	University College Cork, Ireland
Rishab Aiyer Ghosh	University of Maastricht, The Netherlands
Jesus M. Gonzalez-Barahona	University Rey Juan Carlos, Spain
Jean-Luc Hardy	Eurocontrol, France
Jeremy Hayes	University College Cork, Ireland
Joseph Kiniry	University College Dublin, Ireland
Stefan Koch	University of Economics and BA, Austria
Derrick Kourie	University of Pretoria, South Africa
Jean Pierre Laisne	ObjectWeb, France
Karim Lakhani	Harvard, USA
Gregory Lopez	Thales, Frances
Bjorn Lundell	University of Skovde, Sweden
Paul Malone	Waterford Institute of Technology, Ireland
Martin Miclmayr	University of Cambridge, UK
Sandro Morasca	Universita degli Studi dell'Insubria, Italy
Bulent Ozel	Carnegie Mellon University, USA
Witold Pedrycz	University of Alberta, Canada
Paolo Pumilia	EST, Italy
Barbara Russo	University of Bolzano-Bozen, Italy
Gregory Simmons	University of Ballarat, Australia
Marco Scotto	University of Bolzano-Bozen, Italy
Barbara Scozi	Politecnico di Bari, Italy
Katherine Steward	University of Maryland, USA
Malcom Tyrell	Dublin City University, Ireland
Tony Wasserman	Carnegie Mellon University, USA
Hongbo Xu	South China University of Technology, China

Board of Reviewers

List of Contributors

Timo Aaltonen
Tampere University of Technology, Finland
Timo.Aaltonen@tut.fi

Serge Abiteboul
INRIA Futurs, France
Serge.abiteboul@inria.fr

Jyrki Akkanen
Nokia Research Centre, Finland
Jyrki.akkanen@nokia.com

Eileen Allen
Syracuse University, US
eeallen@syr.edu

L. Angelis
Aristotle University, Greece
lef@csd.auth.gr

Maria Antikainen
VTT Technical Research Centre of Finland, Finland
Maria.antikainen@vtt.fi

Claudio Agostina Ardagna
University of Milan, Italy
ardagna@dti.unimi.it

Gabriela Avram
University of Limerick, Ireland
Gabriela.avram@ul.ie

Claudia Ayala
Technical University of Catalunya, Spain
Cayala@lsi.upc.edu

Christian Bac
Groupe des Ecoles de Telecommunications, France
Christianbac@int-evry.fr

Nassim Belbaly
GSCM, France

Hind Benbya
GSCM, France

Evangelia Berdou
London School of Economics and Political Science, UK

Olivier Berger
Groupe des Ecoles de Telecommunications, France
Olivier.Berger@int.evry.fr

Ludger Bischofs
OFFIS, Germany
Ludger.bischofs@offis.de

Patrizia Boccacci
University of Genova, Italy
boccacci@disi.unige.it

Jaap Boender
Universite Paris VII , France
boender@pps.jussieu.fr

Luca Botturi
University of Lugano, Switzerland
botturil@lu.unisi.ch

Ciaran Bryce
University of Geneva, Switzerland
Ciaran.bryce@unige.ch

Michele Cabano
Università di Bologna, Italy
Michele.cabano@unibo.it

Andrea Capiluppi
University of Lincoln, UK
acapiluppi@lincoln.ac.uk

Fabio Caparica
fcl@cin.ufpe.br

Veronica Carrega
University of Genova, Italy
Carrega@disi.unige.it

Scott Christley
University of Notre Dame, USA

Megan Conklin
Elon University, US
mconklin@elon.edu

Reidar Conradi
Norwegian University of Science and Technology, Norway
Conradi@idi.ntnu.no

Grahame S. Cooper
University of Salford, UK
g.s.cooper@salford.ac.uk

Kevin Crowston
Syracuse University, US
Crowston@syr.edu

Jean-Michel Dalle
Université Pierre et Marie Curie, France
Jean-michel.dalle@upmc.fr

Ernesto Damiani
University of Milan, Itlay
damiani@dti.unimi.it

Vincenzo D'Andrea
University of Trento, Italy
dandrea@dit.unitn.it

Itay Dar
Tel Aviv University, Israel
daritay@post.tau.ac.il

Francois Dechelle
EDGE-IT, France
fdechelle@mandriva.fr

Hunor Demeter
Nokia Research Centre, Hungary
Hunor.Demeter@nokia.com

Matthijs den Besten
University of Oxford, UK
Matthijs.denbesten@oerc.ox.ac.uk

Roberto di Cosmo
Universite Paris VII, France
dicosmo@pps.jussieu.fr

Gabriella Dodero
Free University of Bozen, Italy
Gabriella.dodero@unibz.ie

Santiago Duenas
Universidad Rey Juan Carlos, Spain
sduenas@gsyc.escet.urjc.es

Berke Durak
INFRIA Rocqencourt, France
Berke.durak@inria.fr

Tamás Eppel
Budapest University of Technology and Economics, Hungary
peletomi@gmail.com

U. Yeliz Eseryel
Syracuse University, US
Uyeserye@syr.edu

Pat Finnegan
University College Cork, Ireland
P.Finnegan@ucc..ie

Brian Fitzgerald
Queensland University of Technology, Australia
Bf.Fitzgerald@qut.edu.au

Xavier Franch
Technical University of Catalunya, Spain
franch@lsi.upc.edu

Fulvio Frati
University of Milan
Frati@dti.unimi.it

G.R. Gangadharan
University of Trento, Italy
gr@dit.unitn.it

Yongqin Gao
University of Notre Dame, US
Ygao1@nd.edu

Mehmet Gencer
Istanbul Bilgi University, Turkey
Mgencer@cs.bilgi.edu.tr

Alex Sandro Gomes
asg@cin.ufpe.br

Jesus M. Gonzalez-Barahona
Universidad Rey Juan Carlos, Spain
jgb@gsyc.escet.urjc.es

Benoît Hamet
PhpGroupWare Project, France
Benoit.hamet@laposte.net

Helen Hasan
University of Wollongong, Australia
hasan@uow.edu.au

Wilhelm Hasselbring
University of Oldenburg, Germany
hasselbring@informatik.uni-oldenburg.de

Øyvind Hauge
Norwegian University Science and Technology, Norway

Hans-Ludwig Hausen
Fraunhofer, Germany
Hans-Lundwig.Hausen@fit.fraunhofer.de

Robert Heckman
Syracuse University, US
rheckman@syr.edu

Scott Hissam
Carnegie Mellon University, US
shissam@sei.cmu.edu

James Howison
Syracuse University, US
jhowison@syracuse.edu

Francis Hunt
University of Cambridge, UK

Zoltán Ivánfi
Nokia Research Centre, Hungary
Zoltan.ivanfi@nokia.com

Ari Jaaksi
Nokia, Finland
Ari.jaaski@nokia.com

Chris Jensen
Universityof California, US
Cjensen@ics.uci.edu

Jyke Jokinen
Tampere University of Technology, Finland

Uros Jovanvoic
XLAB, Ljubjana, Slovenia
Uros.jovanovic@xlab.si

Stefan Koch
Vienna University of Economics and Business Administration, Austria
Stefan.koch@wu-wien.ac.at

A. Günes Koru
UMBC, US
gkoru@umbc.edu

Jacob Krivoruchka
Nova Southeastern University, US
Krivoruc@nova.edu

Luigi Lavazza
CEFRIEL, Italy
Luigi.lavazza@uninsubria.it

Xavier Leroy
INRIA Rocquencourt, France
Xavier.Leroy@inria.fr

Jingyue Li
Norwegian University of Science and Technology, Norway
jingyue@idi.ntnu.no

Qing Li
Syracuse University, US
Qli03@syr.edu

Juho Lindman
Helsinki School of Economics, Finland
Juho.lindman@hse.fi

Brian Lings
University of Skövde, Sweden
Brian.lings@his.se

Hongfang Liu
Georgetown University Medical Centre
h1224@georgetown.edu

Gregory Lopez
Thales, France
Gregory.lopez@thalesgroup.com

Björn Lundell
University of Skövde, Sweden
Bjorn.lundell@his.se

Greg Madey
University of Notre Dame, US
gmadey@nd.edu

Fabio Mancinelli
Universite Paris VII, France
Fabio@pps.jussieu.fr

Y. Manolopoulos
Aristotlee University, Greece
manolopo@csd.auth.gr

Pentti Marttiin
Nokia, Finland
Pentti.marttiin@nokia.com

Mari Matinlassi
VTT Technical Research Centre of Finland

Riccardo Mazza
University of Lugano, Switzerland
mazzar@lu.unisi.ch

Regis Meissonier
GSCM, France

Janne Merilinna
VTT Technical Research Centre of Finland

Martin Michlmayr
University of Cambridge, UK
Martin@michlmayr.org

Cesare Monti
Università di Bologna, Italy
Cesare.monti@unibo.it

Sandro Morasca
Universita dell'Insubria, Italy
Sandro.morasca@uninsubria.it

Thiago Moreira
tjml@cin.ufpe.br

Lorraine Morgan
University of Limerick, Ireland
Lorraine.morgan@ul.ie

Rogério Nibon
Rtn2@cin.ufpe.br

John Noll
Santa Clara University, US
jnoll@cse.scu.edu

Jukka K. Nurminen
Nokia Research Centre, Finland
Jukka.k.nurminen@nokia.com

Beyza Oba
Istanbul Bilgi University, Turkey
boba@bilgi.edu.tr

Ville Oksanen
Helsinki University of Technology, Finland

Stanislaw Osinski
Poznan Supercomputing and Networking Centre, Poland
Stanislaw.osinski@man.poznan.pl

Bulent Ozel
Carnegie Mellon University, US
bulento@cmu.edu

Michel Pawlak
University of Geneva, Switzerland
pawlak@cui.unige.ch

Mark Perry
University of Western Ontario, Australia
mperry@uwo.ca

Anna Persson
University of Skövde, Sweden
anna.persson@his.se

Clara Pezuela
Atos Origin, Spain
Clara-pezuela@atosorigin.com

Charmaine C Pfaff
University of Wollongong, Australia
Ccp02@uow.edu.au

Giulio Piancastelli
Università di Bologna, Italy
giulio.piancastelli@unibo.it

Radu Pop
INRIA-Mandriva, France
Radu.pop@inria.fr

David Probert
University of Cambridge, UK

Gregorio Robles
Universidad Rey Juan Carlos, Spain
Grex@gsyc.escet.urjc.es

Andreas Rosdal
Norwegian University Science and Technology, Norway

Bruno Rossi
Free University of Bolzano-Bozen, Italy
Bruno.rossi@unibz.it

Christina Rossi
Politecnico di Milano, Italy
Cristina1.rossi@polimi.it

Francesco Rullani
Copenhagen Business School, Denmark
Fr.ivs@cbs.dk

Barbara Russo
Free University of Bolzano-Bozen, Italy
Barbara.russo@unibz.it

Walt Scacchi
University of California, US
wscacchi@ics.uci.edu

Harald Schmidbaueer
Istanbul Bilgi University, Turkey
harald@bilgi.edu.tr

Andrew Schofield
University of Salford, UK
a.j.schofield@pgt.salford.ac.uk

Marko Seppanen
Tampere University of Technology, Finland

Anne Sheehan
University of Limerick, Ireland
Anne.Sheehan@ul.ie

Anders Sigfridsson
University of Limerick, Ireland
Anders.sigfridsson@ul.ie

Alberto Sillitti
Libera Università di Bolzano, Italy
Alberto.sillitti@unibz.it

Darren Skidmore
Monash University, Australia
Darren.skidmore@infotech.monash.edu.au

Sandra A. Slaugher
Carnegie Mellon University, US
Sandras@andrew.cmu.edu

Yuping Song
Henan University of Technology
songyup@gmail.com

Carl-Frederik Sorensen
Norwegian University Science and Technology, Norway

Sulayman K. Sowe
Aristotle University, Greece
sksowe@csd.auth.gr

I. Stamelos
Aristotle University, Greece
stamelos@csd.auth.gr

Petri Stenman
Nokia Ventures Organization, Finland
Petri.stenman@nokia.com

Matthias Studer
University of Geneva, Switzerland
Matthias.studer@metri.unige.ch

Giancarlo Succi
Free University of Bolzano-Bozen, Italy
Giancarlo.succi@unibz.lit

Daniel K. Sullivan
University of Limerick, Ireland
Daniel.Sullivan@ul.ie

Nic Suzor
Queensland University of Technology, Australia
n.suzor@qut.edu.au

Davide Taibi
Universita dell'Insubria, Italy
Davide.taibi@uninsubria.it

Stefano Tardini
University of Lugano, Switzerland
Tardinis@lu.unisi.ch

Ralf Treinen
LSV, ENS de Cachan, France
treinen@lsv.ens-cachan.fr

Vehbi Sinan Tunalio Lu
Istanbul Bilgi University, Turkey
Vst@cs.bilgi.edu.tr

Tere Vaden
University of Tampere, Finland

Niklas Vainio
University of Tampere, Finland

Jaani Väisänen
Tampere University of Technology, Finland
Jaani.vaisanen@tut.fi

Frank van der Linden
Philips Medical Systems, The Netherlands
Frank.van.der.linden@phillips.com

Manon van Leeuwen
FUNDECYT, Spain
manon@fundecyt.es

Gabriel Vasile
INRIA Futurs, France
gabriel.vasile@inria.fr

Dan Vodislav
CEDRIC-CNAM, France
vodislav@cnam.fr

Jérôme Vouillon
Universite Paris VII, France
vouillon@pps.jussieu.fr

Quang Vu Dang
Group des Ecoles de Telecommunications, France
VuDang-Quang@int.evry.fr

Dawid Weiss
Poznan University of Technology, Poland
Dawid.weiss@cs.put.poznan.pl

Michael Weiss
Carleton University, Canada
Weiss@scs.carelton.ca

Dongsong Zhang
UMBC, US
zhangd@umbc.edu

Contents

Part I

Full Papers

FOCSE: An OWA-based Evaluation Framework for OS Adoption in Critical Environments

Claudio Agostino Ardagna, Ernesto Damiani, Fulvio Frati
University of Milan - via Bramante 65, Crema (CR), Italy
{ardagna,damiani,frati}@dti.unimi.it

Abstract. While the vast majority of European and US companies increasingly use open source software for non-key applications, a much smaller number of companies have deployed it in critical areas such as security and access control. This is partly due to residual difficulties in performing and documenting the selection process of open source solutions. In this paper we describe the FOCSE metrics framework, supporting a specific selection process for security-related open source code. FOCSE is based on a set of general purpose metrics suitable for evaluating open source frameworks in general; however, it includes some specific metrics expressing security solutions' capability of responding to continuous change in threats. We show FOCSE at work in two use cases about selecting two different types of security-related open source solutions, i.e. *Single Sign-On* and *Secure Shell* applications.

1 Introduction

In the last decade, open source operating systems and middleware platforms have been widely deployed [4]. In the security area, the adoption of open source solutions has been much slower, since most users do not completely trust the open source community and consider open source middleware a potential "backdoor" for attackers, potentially affecting overall system security. However, proprietary security solutions have their own drawbacks such as vendor lock-in, interoperability limitations, and lack of flexibility. Recent research suggests that the open source approach can overcome these limitations [3, 21]. According to some researchers, open source solutions may even in the end improve security, as they give greater visibility of software vulnerabilities [11], giving the possibility to fix them as soon as a threat is described. In our opinion, what is still missing to boost open source adoption in security is a selection framework allowing the users to evaluate the level of suitability of different open source security solutions. In itself, comparative evaluation of OSS is a time-honored subject, and several researchers [8, 12] have proposed complex methodologies dealing with the evaluation of open source

Please use the following format when citing this chapter:

Ardagna, C.A., Damiani, E., Frati, F., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 3–16.

products from different perspectives, such as code quality, development flow and community composition and participation. General-purpose open source evaluation models, such as Bernard Golden's *Open Source Maturity Model* (OSMM) [14] do not address some specific requirements of security software selection. However, these models assess open source products based on their maturity, i.e. their present production-readiness, while evaluating security solutions also involves trying to predict how fast (and how well) a security component will keep abreast of new attacks and threats. A security-oriented software evaluation framework should provide potential adopters with a way to compare open source solutions identifying the one which *i)* best suits their current non-functional requirements and *ii)* is likely to evolve fast enough to counter emerging threats.

In this paper, we develop on our previous work [5] to obtain a specific technique for evaluating open source security software, including access control and authentication systems. Namely, we describe a *Framework for OS Critical Systems Evaluation* (FOCSE) based on a set of metrics aimed at supporting open source evaluation, through a formal process of adoption. FOCSE evaluates an open source project in its entirety, assessing the community composition, responsiveness to change, software quality, user support, and so forth.

The remainder of this paper is organized as follows. After a brief introduction of the basic concepts of comparative software evaluation (Section 2) we present our set of evaluation metrics (Section 3). Then, Section 4 presents the aggregator used to integrate different metrics in a single estimation, allowing for ranking analyzed solutions. Finally, Section 5 provides two use cases where the defined framework evaluates open source Single Sign-On (SSO) and Secure Shell (SSH) solutions. Section 6 gives our conclusions.

2 Basic Concepts

In this section, we provide a review of the technologies used in the context of FOCSE evaluation. In particular, we describe the `FLOSSmole` project [13] used to gather and store data about the open source solutions to be evaluated. An essential prerequisite of FOCSE is the availability of the raw data necessary to compute the metrics defined in Section 3. The availability of a database can greatly improve the reliability and the effectiveness of FOCSE. FLOSSmole (formerly OSSmole) [10,13] is a platform designed to gather, share and store data supporting comparative analysis of open source software. FLOSSmole is aimed at: *i)* collecting raw data about open source projects; *ii)* providing summary reports about open source projects; *iii)* integrate data from other research teams; *iv)* provide tools to gather the data of interest. In the following analysis, we relied on FLOSSmole for collecting information about the projects subject to our analysis.[1]

3 FOCSE: a Framework for OS Critical Systems Evaluation

Generally speaking, few organizations rely on internal guidelines for the selection of open source products. Our experience has shown that in most cases project leaders select an open source solution based on its being readily available and fulfilling their functional requirements [5, 6]. FOCSE evaluation criteria are aimed at evaluating each open source project in its entirety, highlighting the promptness of reacting against newly discovered vulnerabilities or incidents. Applications success, in fact, depends on the above principle because a low reaction rate to new vulnerabilities or incidents implies higher risk for users that adopt the software, potentially causing loss of information and money.

3.1 Evaluation principles

FOCSE evaluation is based on six macro-areas [5]:

- *Generic Aspects (GA).* i.e. all quantitative attributes expressing the solution's non-functional features, i.e. those not related to its purpose or scope (for a complete list, see [8]).
- *Developers Community (DC).* i.e. quantitative attributes expressing the composition and diversity of the developers community. A high number of developers from different countries allows sharing of diverse backgrounds and skills, giving vitality and freshness to the community and helping in solving problems, including bugs definition and fixing.
- *Users Community (UC).* The success of an open source application can be measured in terms of number and profile of the users that adopt it and rely on it. Obviously, measuring and evaluating the users community is less simple than doing so for developers because users interacting with an open source project are often anonymous. The users community, however, can be quantitatively estimated by means of parameters like the number of downloads, the number of requests, the number of posts inside the forum, and the number of users subscribed to the mailing list. A qualitative measure of this macro-area can also be given by the profile of the users adopting the project: if users belong to well-known companies or organizations and report positive results, the solution's UC indicators can be enhanced.

[1] Of course, FOCSE does not mandate the use of FLOSSmole, as data can be gathered manually by the evaluator. However, companies interested in comparative evaluation of OSS solutions should rely on a certified and repeatable data collection technique.

- *Software Quality (SQ)*. This area include metrics of quality built into the software by the requirements, design, code and verification processes to ensure that reliability, maintainability, and other quality factors are met.[2]
- *Documentation and Interaction support (DIS)*. This macro area is composed of two major sub-areas: *traditional documentation,* that explains the characteristics, functionalities and peculiarities of the software and *support,* in terms of time allotted by developers to give feedback via forums, mailing lists, white papers, and presentations.
- *Integration and Adaptability with new and existing technologies (IA)*. A fundamental tenet of OS projects is full integration with existing technologies at project startup and a high level of adaptability to new technologies presented during project life. Another fundamental aspect is the ability of the developers' community to solve and fix bugs and react to new vulnerabilities.

3.2 Evaluation parameters

We now provide the detailed description of some metrics (see Table 1 and 2) and their distribution in the six macro-areas described above. These metrics are then used (Section 5) to evaluate and compare open source security applications. Regardless of the macro-area they belong to, our quantitative metrics are orthogonally divided in: *i) Core Metrics (CM)*, and *ii) Advanced Metrics (AM)*.

Core Metrics

Core Metrics include all metrics that can be readily computed from current information on the projects. These metrics are based on data that can be usually found in the projects web sites; however structured data sources like FLOSSmole [13] can make the evaluation process stronger and more trustworthy.

- *Age*, that represents the age in days of the project, calculated from the date of the first stable release.
- *Last Update Age*, that represents the age in days of the last stable project update. It is calculated as the difference between the date of the last update and the current date. Differently from *Age* metric, *Last Update Age* measures the level of freshness of the last application update, and it allows the identification of dead projects.
- *Project Core Group*, a boolean value that evaluates the existence of a stable core group of developers who have been working on the project from its inception (or for at least three-quarters of its Age). Core developers are

[2] Open source security solutions lend themselves to quality assurance and evaluation based on shared testing and code walk-through as outlined in [1]. However, comparing reference implementations of security solutions based on code walk-through is outside the scope of this paper.

defined as the ones that contribute both to project management and code implementation.

- *Number of Core Developers*, strictly related to the above Project Core Group metric, measures the number of core developers.
- *Number of Releases*, that measures the number of releases from the project start up. A high number of releases could indicate the vitality of the community and its promptness on reacting against new threats and vulnerabilities.

Table 1. Evaluation Metrics Definition: Core Metrics

Core Metrics			
Name	**Definition**	**Values**	**Area**
Age	Age of the project.	Days	GA
Last Update Age	Age of the last project update.	Days	GA
Project Core Group	Evaluate the existence of a group of core developers.	Boolean	GA,DC
Number of Core Developers	Number of core developers contributing the project.	Integer	DC
Number of Releases	Number of releases since project start up.	Integer	SQ,IA
Bug Fixing Rate	Rate of bugs fixed.	Real	SQ,IA
Update Average Time	Vitality of developers group, i.e. mean number of days to wait for a new update (release or patch).	Days	SQ,IA
Forum and Mailing List Support	Check forum and mailing list availability.	Boolean	GA,DIS
RSS Support	Check RSS availability.	Days	GA,DIS
Number of Users	Number of users adopting the application.	Integer	UC
Documentation Level	Level of project documentation, in terms of API, user manuals, whitepapers.	Mbyte	DIS
Code Quality	Qualitative measure of code quality.		SQ,IA
Community Vitality	Vitality of the community in terms of number of forum threads and replies.	Real	DC,UC

- *Bug Fixing Rate*, which measures the rate of bugs fixed looking at bugs and fixings reports of each product. To prevent young projects with few bugs fixed from outperforming old and stable projects with hundreds of bugs fixed, the bug fixing rate is weighted over the total number of bugs detected. This rate is computed as:
$$\left(\#ofBugsFixed + 1/\#ofBugsDetected + 1\right)\left(1 - e^{-\#ofBugsDetected}\right).$$
We stress the fact that this metrics is available from well known security-related sources, such as the Computer Emergency Response Team (CERT) [9], providing detailed information about discovered bugs.

- *Update Average Time*, measuring the vitality of an open source community. It indicates the average number of days between releases of major and minor versions (patches) of the product. This metric is calculated as: *age/(#ofPatches + #ofReleases).*
- *Forum and Mailing List Support*, a boolean value expressing availability of forum and mailing lists at the products' web sites. This is an important feature of open source products since it hints at a strict collaboration between users and developers communities.
- *RSS Support*, a boolean value expressing availability of RSS (Really Simple Syndication), i.e., a family of web feed formats, used to publish frequently updated digital content. This is an important feature of open source products since it allows users who download and rely on a particular project to be fully informed of the project news.
- *Number of Users*, expressing the number of users adopting the product; this value can be roughly approximated as: *#ofDownloads/#ofReleases*. This parameter is also an indicator of the product's popularity.
- *Documentation Level*, expressing the documentation level (in Mbyte) in terms of APIs documentation, user manuals, whitepapers, and so on.
- *Code Quality*, that measures the intrinsic quality of the software product.[3]
- *Community Vitality*, that measures the vitality of the community in term of answers given in the forum in response to specific users questions. This value is computed as: *#ofForumReplies/#ofForumThreads*.

The core metrics set is summarized in Table 1.

Advanced Metrics

Advanced Metrics include evaluation parameters requiring privileged access to the developers' community [5]. Otherwise, they can be estimated basing on raw data**.**

- *Group/Developers Stability*, that measures the degree of stability of developers group and, consequently, the stability of the product itself. Each developer is classified as *stable* or *transient*, where stable is a developer that continuously contributes to code, transient in the other cases. The exact number of contributions to make a developer stable is project-dependent. This value is computed as: *#ofStableDevelopers/#ofDevelopers*100.*
- *Project Reputation*, which estimates the reputation of the project by aggregating the evaluations provided by the project developers and users. Several algorithms for assessing reputation are available [16].
- *Repository Quality*, that provides an estimation of the repository where project is hosted.[4] It can be computed in several different ways; we chose to

[3] This metric is included for completeness, but its measurement is out of the scope of this paper (see Section 3.1).

[4] For self-hosted projects this metrics is set to 0.

measure it as the number of active projects hosted by the repository (an active project is defined as one that has released at least an update within a year), over the total number of hosted projects: *#ofActiveProjects/#ofProjects*.

- *Reaction Rate*, that estimates the average time the developers community takes to find solutions to newly discovered vulnerabilities. This parameter measures the community's promptness in terms of reaction against discovered software vulnerabilities.
  Given V as the set of vulnerabilities, this metric is defined as:
  $$\left(n * UpdateAverageTime\right) \Big/ \sum_{i=1}^{n} FixingDate\left(V_i\right) - DiscoveringDate\left(V_i\right)$$
  where $V_i \in V$ and $n = \left|V\right|$.
- *Incident Frequency*, that measures the robustness of the application with respect to newly discovered vulnerabilities. This parameter is computed as *#ofIncidents*/ *|V|* where *V* is the set of vulnerabilities.

Table 2. Evaluation Metrics Definition: Advanced Metrics

Advanced Metrics			
Name	**Definition**	**Values**	**Area**
Group/Developers Stability	Measures the degree of stability of a developers group.	[0..100%]	DC
Reaction Rate	Average time needed by the developers' community to find solutions for newly discovered vulnerabilities. More specifically, it represents the project developers' ability in reacting to the set of vulnerabilities.		IA
Repository Quality	Estimator of the project repository quality.		GA
Incident Frequency	Measures the number of incidents due to vulnerabilities.		IA
Project Reputation	Measure the project reputation by aggregating the evaluation provided by project developers and users.		GA

The first metrics, Groups/Developers Stability, is not easy to estimate from outside the developers' community. It may be however available to insiders, e.g. to companies that adopt an open source product and actively contribute to its community. Finally, regarding the computation of the last two parameters, we stress the fact that various security-related Web portals provide databases that contain information about vulnerabilities and related incidents summaries. In particular, three main portals stand out: *Secunia* (http://secunia.com/) that offers monitoring of vulnerabilities in more than 12,000 products, *Open Source*

Vulnerability Database (OSVDB) (http://www.osvdb.org/) an independent database that provides technical information about vulnerabilities and, finally, CERT, which provides a database containing information about vulnerabilities, incidents and fixes.

In summary, most of the information required to compute FOCSE advanced metrics is already available on the Net. Unfortunately, this information being in raw format makes it difficult to automate the computation, as substantial pre-processing is needed to compute these metrics.

4 Aggregating Heterogeneous Results

To generate a single estimation, it is necessary to aggregate the metrics values. This way, two or more projects, each one described by its set of attributes, can be ranked by looking at their FOCSE estimations. Below, the *Ordered Weighted Average* (OWA) operator, used to aggregate the defined metrics, is introduced.

4.1 OWA Operator

Ordered Weighted Averaging (OWA) operators, originally introduced by Ronald Yager [24, 26], provide a parameterized family of mean-type aggregation operators. An important feature of these operators is the *reordering* step, which makes OWA a nonlinear operator. OWA operator is different from the classical weighted average in that coefficients are not associated directly with a particular attribute but rather to an ordered position. The structure of these operators is aimed at combining the criteria under the guidance of a quantifier.

Definition 1 (OWA Operator) *Let* $w = [\omega_1, \omega_2, \ldots, \omega_n]$ *a weight vector of dimension* n*, such that* $\omega_i \in [0,1]$ *and* $\sum_i \omega_i = 1$. *A mapping* $f_{OWA} : R^n \longrightarrow R$ *is an OWA operator of dimension* n *if*

$$f_{OWA}(a_1, a_2, \ldots, a_n) = \sum_i \omega_i a_{\sigma(i)}$$

where $\{\sigma(1), \ldots, \sigma(n)\}$ *is a permutation of {1,...,n} such that* $a_{\sigma(i)} \le a_{\sigma(i-1)}$ *for i=2,…,n.*

We adopt the monotonic quantifiers Q_{mean} [26]. The pure averaging quantifier has w_j=1/n for all j=1,…,n having Q_{mean}(K)=K/n as its linear quantifier.

The previous quantifier represents the set of weights used in our experimentation (i.e., [1/n, 2/n,…,(n-1)/n, 1]). All decision process involving multiple criteria like software selection involve some compensatory trade-offs. Trade-offs occurs in the presence of conflicting goals, when compensation between the corresponding compatibilities is allowed. OWA operators can realize trade-offs between objectives, by allowing a positive compensation between ratings, i.e. a higher degree of satisfaction of one of the criteria can compensate for a lower degree of satisfaction of other criteria to a certain extent. OWA operators provide for any level of compensation lying between logical conjunction and disjunction. An interesting feature of OWAs is their adaptability. To any specific software selection problem we can tailor an appropriate OWA aggregation operator from some rules and/or samples determined by the decision makers.

5 Applying FOCSE to Existing Critical Application

We introduce two categories of open source security solutions: SSO systems and SSH clients. Then, we show how selection can be made by first evaluating the FOCSE metrics, and then by aggregating them by means of OWA operator.

5.1 Single Sign-On Frameworks

The SSO [15] approach is aimed at co-ordinating and integrating user log-on mechanisms of different domains. In particular, SSO provides a technique where a primary domain is responsible for managing all user credentials and information used during the authentication process, both to the primary domain itself and to each of the secondary domains that the user may potentially require to interact with. SSO also provides the users with a transparent authentication to the secondary domains. In this scenario, the following subset of SSO frameworks has been evaluated by FOCSE metrics (for more details, see [2]).

- *Central Authentication Service*. Central Authentication Service (CAS) [7] is an open source authentication system originally developed at Yale University. It implements a SSO mechanism aimed at providing a *Centralized Authentication* to a single server through *HTTP redirections*.
- *SourceID*. SourceID [22], first released in 2001 by Ping Identity Corporation, is an open source multi-protocol project for enabling identity federation and cross-boundary security. SourceID also implements Liberty Alliance SSO specifications.
- *Java Open Single Sign-On (JOSSO)*. Java Open Single Sign-On (JOSSO) is an open source SSO infrastructure based on J2EE specifications. In detail, JOSSO provides a solution for centralized platform-neutral user authentication [17], combining several authentication schemes (e.g., username/password or certificate-based) and credential stores.
- *Open Source Web SSO*. The Open Source Web SSO (Open SSO) [18] project relies on the consolidated Web SSO framework developed by Sun

Microsystems, that was opened to the open source community in July 2005. It provides services and functionalities for implementing transparent SSO as an infrastructure security component.

5.2 SSH Clients

SSH is a communication protocol widely adopted in the Internet environment that provides important services like secure login connections, file transfers and secure forwarding of X11 connections [27]. SSH protocol allows also a communication approach named Tunneling as a way to encapsulate a generic communication flow in SSH packets, implementing a port forwarding mechanism and securing data that use untrusted communication protocols, exploiting SSH encryption features. The FOCSE framework has been applied for evaluating the following SSH clients.

- *Putty*. Putty [20] is a popular open source SSH client for Microsoft Windows platforms. It supports versions 1 and 2 of SSH protocol, terminal emulation, and provides a complete and essential user interface.
- *WinSCP*. WinSCP [25] allows safe copying of files between remote internet machines through the SSH protocol. It also offers basic remote management operations, such as file duplication, renaming and deleting, and supports all the encryption features of SSH protocol.
- *ClusterSSH*. ClusterSSH [23] allows users to open and manage several terminal windows with connections to specified hosts and an administration console. The tool is also intended for cluster administration.

5.3 SSO Comparison

Table 3 gives a comparison of SSO implementations based on FOCSE metrics.[5] Focusing on evaluation, as shown by Table 3, all systems are quite stable due to the fact that their start-up happened more than a year ago. Even Open SSO, i.e. the most recent one, can be considered as a stable implementation since it represents an open source extension of a well-established proprietary implementation, named *Sun Java System Access Manager*. A common characteristic shared by all analyzed solutions is that they are managed by a consolidated core group providing stability to the project and coordination to open source community. By contrast, these solutions have different documentation levels. Specifically, whereas CAS provides a good amount of documentation, i.e. `28.55 MB`, SourceID presents on its Web sites only a limited amount of information, i.e. `8.96 MB`. Although at the first sight the number of releases could seem a good estimation of projects vitality, this is not entirely true. Often, in fact, the number of releases is highly dependent on the project age. To the

[5] Due to the fact that only JOSSO and Open SSO data have been gathered by FLOSSmole, our evaluation is sensitive to errors due to obsolete information.

purpose of clearly evaluating the liveness of a project, the number of releases should be coupled with the Update Average Time. In particular, Table 3 seems to suggest that Open SSO is the liveliest project. However, its low update average time is due to the fixing of youth problems that hints to keep Open SSO out of this metric comparison. We argue, then, that the more active and viable implementation is JOSSO, because it provides a new release every `44 days`. Also, the Bug Fixing Rate metric suggests that JOSSO is the most reactive project between the analyzed solutions. Concerning *Last Update Time*, CAS implementation achieves the best results, i.e. `18 days`. Also, CAS is the only project providing a certified list of users. Here we do not consider the Number of Users, Repository Quality, and Community Vitality parameters for all solutions, because relatively few solutions provide enough information to clearly and unambiguously compute them. In particular, CAS provides a certified list of the CAS' deployers, and JOSSO and Open SSO make possible to easily compute the community vitality.

Table 3. Comparison of proposed SSO implementations at 31 December 2006.

Metrics	CAS	SourceID	JOSSO	Open SSO
Age (GA)	1865 days	1177 days	854 days	570 days
Last Update Age (GA)	18 days	236 days	217 days	21 days
Project Core Group (GA,DC)	Yes	Yes	Yes	Yes
Number of Core Developers (DC)	5	N/A	2	N/A
Number of Releases (SQ,IA)	28	7	7	1 (since code opening)
Bug Fixing Rate (SQ,IA)	N/A	N/A	0.78	0.53
Update Average Time (SQ,IA)	67 days	168 days	44 days	27 days
Forum and Mailing List Support (GA,DIS)	Mailing List Only	Mailing List Only	Yes	Yes
RSS Support (GA,DIS)	Yes	Yes	No	Yes
Number of Users (UC)	48 (certified)	N/A	7072 (approx.)	N/A
Documentation Level (DIS)	28.55 MB	8.96 MB	16.96 MB	14.3 MB
Community Vitality (DC,UC)	N/A	N/A	1.87	3.56

5.4 SSH Comparison

Table 4 gives a comparison of SSH client implementations. Differently from SSO systems, all the analyzed SSH frameworks lie in FLOSSmole database.

Table 4. Comparison of proposed SSH implementations at 31 December 2006.

Metrics	Putty	SourceID	JOSSO
Age (GA)	2911 days	1470 days	1582 days
Last Update Age (GA)	636 days	238 days	159 days
Project Core Group (GA,DC)	Yes	Yes	Yes

Number of Core Developers (DC)	4	2	2
Number of Releases (SQ,IA)	15	32	15
Bug Fixing Rate (SQ,IA)	0.67	N/A	0.85
Update Average Time (SQ,IA)	194 days	46 days	105 days
Forum and Mailing List Support (GA,DIS)	N/A	Forum Only	Yes
RSS Support (GA,DIS)	No	Yes	Yes
Number of Users (UC)	N/A	344k	927
Documentation Level (DIS)	1.39 MB	10 MB	N/A
Community Vitality (DC,UC)	N/A	3.73	5.72

Focusing on the evaluation, it is clear that all the projects are stable since their startup happens more than four years ago. This results in stable and consolidated project core groups of at least two core developers, and in a good number of releases. Concerning the *Bug Fixing Rate* metric, whereas for WinSCP no data are available, Putty and ClusterSSH provide a good bug fixing rate, 0.67 and 0.85 respectively.

To conclude, the number of users adopting WinSCP (i.e., 344K of users) is impressive, suggesting that it is very attractive for users to take advantage of open source SSH solutions.

5.5 Applying OWA Operator to FOCSE Critical Application Comparison

We now apply OWA operator to provide a single estimation of each evaluated solution. For the sake of conciseness, we shall only show the details of OWA application to CAS solution. All other solutions can be processed in the same vein.

First, the adoption of OWA operator together with the Q_{mean} quantifier results in the following set of weights:

$$w=[\tfrac{1}{12},\tfrac{2}{12},\tfrac{3}{12},\tfrac{4}{12},\tfrac{5}{12},\tfrac{6}{12},\tfrac{7}{12},\tfrac{8}{12},\tfrac{9}{12},\tfrac{10}{12},\tfrac{11}{12},1]$$

In particular, the Q_{mean} identifier is used to mitigate the impact of too high and too low values on the overall aggregation process.[6] Then, after normalizing the vector of weights:

$$w_n=w/\tfrac{78}{12}=[\tfrac{1}{78},\tfrac{1}{39},\tfrac{1}{26},\tfrac{2}{39},\tfrac{5}{78},\tfrac{1}{13},\tfrac{7}{78},\tfrac{4}{39},\tfrac{3}{26},\tfrac{5}{39},\tfrac{11}{78},\tfrac{2}{13}]$$

we normalized the vector of CAS attributes as $attrValue_k/maxAttrValue$, where $attrValue_k$ is the value of the k-th attribute and *maxAttrValue* is the maximum attribute value among all the analyzed solutions. It is important to underline that attributes such as *Last Update Age*, where low values mean better evaluations, are normalized as 1- $attrValue_k/maxAttrValue$. The normalization process results in the ordered vector a=[1, 1, 1, 1, 1, 1, 0.93, 0.5, 0.4, 0.01, 0, 0].

Now, we calculate the final value of CAS system as: $f_{OWA}(a)=\sum_{i=1}^{12} w_i a_i = 0.45$.

[6] Different quantifiers could be adopted depending on the scenario.

When the same process is applied to all solutions, one obtains the two tables depicted in Table 5. To conclude, from Table 5 is clear that whereas among SSO systems the best solution is CAS, followed by JOSSO implementation, concerning SSH clients, the solution more likely to be adopted is WinSCP.

Table 5. OWA-based Comparison.

(a) SSO Comparison

	CAS	**SourceID**	**JOSSO**	**Open SSO**
f_{OWA}	0.45	0.19	0.36	0.34

(b) SSH Comparison

	Putty	**SourceID**	**JOSSO**
f_{OWA}	0.23	0.51	0.47

6 Conclusions and Future Work

We presented our FOCSE framework aimed at the definition of a quantitative approach to the comparative evaluation of security-related open source systems. A structured set of metrics used in the evaluation process and specifically designed for such systems, a formal aggregation is introduced to deal with the heterogeneity of such metrics. This aggregation allows the FOCSE evaluation to be expressed by means of a single value and to be more user-friendly. Then as case-studies, we compared some well-known implementations of SSO and SSH applications. Future work will study the integration of FLOSSmole-like databases in FOCSE, allowing the definition of an infrastructure able to gather the requested data by itself and then provide the evaluation in a transparent way to the user. Also the definition of a validation system of open source projects based on community inputs [19], as well as the definition of an extended version of the framework able to evaluate whatever open source solution will be subject of future research.

Acknowledgements

This work was supported in part by the European Union within the PRIME Project in the FP6/IST Programme under contract IST-2002-507591, and by the by the Italian Ministry of Research under FIRB contracts n. RBNE05FKZ2_004 TEKNE and n. RBNE01JRK8_003 MAPS.

References

1. S. Abiteboul, X. Leroy, B. Vrdoljak, R. Di Cosmo, S. Fermigier, S. Lauriere, F. Lepied, R. Pop, F. Villard, J.P. Smets, C. Bryce, K.R. Dittrich, T. Milo, A. Sagi, Y. Shtossel, and E.

Panto. Edos: Environment for the development and distribution of open source software. In *Proc of The First International Conference on Open Source Systems*, pages 66–70, Genova (Italy), July 2005.

2. C.A. Ardagna, E. Damiani, S. De Capitani di Vimercati, F. Frati, and P. Samarati. CAS++: an open source single sign-on solution for secure e-services. In *Proc. of the 21st IFIP International Information Security Conference "Security and Privacy in Dynamic Environments"*, May 2006.
3. C.A. Ardagna, E. Damiani, F. Frati, and M. Madravio. Open source solution to secure e-government services. *Encyclopedia of Digital Government*, 2006.
4. C.A. Ardagna, E. Damiani, F. Frati, and M. Montel. Using open source middleware for securing e-gov applications. *In Proc. of The First International Conference on Open Source Systems*, pages 172–178, Genova (Italy), July 2005.
5. C.A. Ardagna, E. Damiani, F. Frati, and S. Reale. Adopting open source for mission-critical applications: A case study on single sign-on. In *Proc. of IFIP Working Group 2.13 Foundation on Open Source Software*, volume 203/2006, pages 209–220, Como, Italy, 2006.
6. C.A. Ardagna, E. Damiani, F. Frati, and S. Reale. Secure authentication process for high sensitive data e-services: A roadmap. *Journal of Cases on Information Technology*, 9(1):20–35, 2007.
7. P. Aubry, V. Mathieu, and J. Marchal. Esup-portal: open source single sign-on with cas (central authentication service*). In Proc. of EUNIS04 – IT Innovation in a Changing World*, pages 172–178, Bled (Slovenia), 2005.
8. A. Capiluppi, P. Lago, and M. Morisio. Characteristics of open source projects. In *CSMR*, page 317, 2003.
9. CERT-CC. Cert coordination center. Available: www.cert.org/.
10. M. Conklin. Beyond low-hanging fruit: Seeking the next generation in floss data mining. In *Proc. of IFIP Working Group 2.13 Foundation on Open Source Software*, volume 203/2006, Como, Italy, 2006.
11. C. Cowan. Software security for open-source systems. *IEEE-SEC-PRIV*, 1(1):38–45, January/February 2003.
12. J. Feller and B. Fitzgerald. A framework analysis of the open source software development paradigm. In *ICIS*, pages 58–69, 2000.
13. FLOSSmole. Collaborative collection and analysis of free/libre/open source project data. Available: ossmole.sourceforge.net/.
14. B. Golden. Succeeding with Open Source. *Addison-Wesley Professional*, 2004.
15. The Open Group. Single sign-on. Available: www.opengroup.org/security/sso/.
16. A. Josang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. In *Decision Support Systems*, 2005.
17. JOSSO. Java open single sign-on. Available: sourceforge.net/projects/josso.
18. OpenSSO. Open web SSO. Available: opensso.dev.java.net/.
19. E. Damiani P. Ceravolo and M. Viviani. Bottom-up extraction and trust-based refinement of ontology metadata. *IEEE Transaction on Knowledge and Data Engineering*, 19(2):149–163, February 2007.
20. PuTTY. A free telnet/ssh client. Available: www.chiark.greenend.org.uk/~sgtatham/putty/.
21. E.S. Raymond. The cathedral and the bazaar. Available: www.openresources.com/documents/cathedral-bazaar/, August 1998.
22. SourceID. Open source federated identity management. Available: www.sourceid.org/.
23. Cluster SSH. Cluster admin via ssh. Available: sourceforge.net/projects/clusterssh.
24. V. Torra. The weighted OWA operator. *International Journal of Intelligent Systems*, 12(2):153–166, 1997.
25. WinSCP. Free sftp and scp client for windows. Available: winscp.net/eng/index.php.
26. R.R. Yager. On ordered weighted averaging aggregation operators in multi-criteria decision making. *IEEE Transaction Systems, Man, Cybernetics*, 18(1):183–190, January/February 1988.
27. T. Ylonen. Ssh - secure login connections over the internet. In *Proc. of Sixth USENIX Security Symposium*, page 3742, San Jose, California, USA, 1996.

Open Source Collaboration for Fostering Off-The-Shelf Components Selection

Claudia Ayala[1], Carl-Fredrik Sørensen[2], Reidar Conradi[2], Xavier Franch[1], Jingyue Li[2]
1 Technical University Of Catalunya (UPC), Software Department
Campus Nord- Omega Building. Barcelona, Spain
{cayala,franch}@lsi.upc.edu
WWW home page: http://www.lsi.upc.es/~webgessi/index.html
2 Norwegian University of Science and Technology (NTNU)
Department of Computer and Information Science
NO-7491, Trondheim, Norway
{carlfrs,conradi,jingyue}@idi.ntnu.no
WWW home page: http://www.idi.ntnu.no/grupper/su/

Abstract. The use of Off-The-Shelf software components in Component-Based Development implies many challenges. One of them is the lack of available and well-suited data to support selection of suitable OTS components. This paper proposes a feasible and incremental way to federate and reuse the different efforts for finding, selecting, and maintaining OTS components in a structured way. This is done not only for supporting OTS components selection, but also to overcome reported problems with the integration and maintenance of component repositories. It is based on the "open source collaboration" idea to incrementally build an OTS components reuse infrastructure, enabling automatic support for OTS selection processes.

Keywords: Off-the-Shelf components (OTS), Commercial-Off-The-Shelf (COTS), Open Source Software (OSS), open source collaboration, component selection, reuse.

1. Introduction

The use of Off-The-Shelf software components –hereafter OTS- as part of large software systems have grown steadily [1]. Consequently, a huge amount of OTS has become accessible in the market. OTS mainly come in two major kinds: COTS (Commercial-Off-The-Shelf) owned by commercial vendors that often provide specific support [1]; and OSS (Open-Source-Software) provided by open source communities with freely accessible source code, but with no promise of specific

Please use the following format when citing this chapter:

Ayala, C., Sørensen, C.-F., Conradi, R., Franch, X., Li, J., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 17–30.

support [2]. Especially in the latter case, software engineering researchers and practitioners have become increasingly aware of the contribution that open source development is offering to the software industry, business, and society in general [3].

An OTS is defined as: "*a software product that is publicly available at some cost or with some licensing obligations and other software projects can reuse and integrate it into their own products*" [4]. The selection of OTS has been recognized as a critical process in the OTS-based development risk mitigation [5]. Regardless the specific properties of OSS and COTS components, we may consider that the high-level selection process (i.e. the practice of locating candidates, evaluating them with respect to the system requirements, and making the final choice) is essentially similar.

From our empirical studies aimed to investigate the state-of-the-practice in industrial OTS selection projects, we realized that their success is highly dependent on the quality and completeness of the data available concerning these components. Currently, these data are highly heterogeneous, since it comes from different sources, and sometimes its trustworthiness is unclear. Also, in those contexts (consultant companies, some IT departments, etc.) that select OTS periodically, the reuse of these data would improve the effectiveness of the selection processes. Heterogeneity and lack of reuse have a negative effect on the perceived risks of using OTS for integrating large industrial systems [6]. In fact, it is considered a major challenge for fostering the adoption of OTS in industrial frameworks, especially for OSS components [7]. This problem is even more evident for OSS components given their free and collaborative development nature that lacks of a structured documentation and a marketing channel behind. In this paper, we describe a feasible and incremental way to federate and reuse the actual efforts for selecting, and maintaining OTS in a structured way. We propose a Wiki-based portal based on a flexible metamodel that enables people (e.g., research groups, individuals or organizations) to work collectively in an open-source-like environment for obtaining, sharing, managing, storing, retrieving, and reusing OTS information for supporting the (re)use of OTS.

2. State-of-the-Art and State-of-the-Practice

2.1 Industrial Practice

Some empirical studies in companies using OTS show several relevant results about how they select and use such components [8]. Such studies reveal that they do not normally use any formal process for selecting components. Instead, most of them are using an experience-based and/or hands-on trial-based selection processes. In the first case, developers already have experience with some specific components or technology, and this experience is important in deciding which components to choose. In the second case, the World Wide Web is used to find executable components and a few of them are then downloaded and further evaluated. Additionally, such studies also demonstrated that OSS components are rarely modified, but used and integrated as is. Based on these empirical studies, we highlight in Table 1 the high-level activities directly related to the OTS selection processes once the decision to acquire OTS is made. These activities are not intended to fully describe OTS selection processes; but provide a general description with the goal of identifying the most relevant required roles. These roles are informal and implicit played by the respondents (i.e. they were not explicitly established in the actual practice). We also observe that in the case of organizations that continuously perform OTS selection processes, it is important to reuse their knowledge about the components and decisions taken. Thus, we envisage the *Knowledge Keeper* role, even when an explicit documentation is not formally written, but existing as tacit knowledge in the head of the involved people [8].

Table 1. Activities and Roles in OTS Selection

Activity	OTS Users Role
Finding Candidates OTS	*Market Watcher (MW)* explores the marketplace segments to find components that may match the established requirements.
Evaluating OTS Candidates	*Quality Engineer (QE)* measures the factors that are related to the requirements in the candidate components.
Deciding OTS Component	*Selector (S)* takes the final decision based on the evaluation of the candidates and also taking into account other relevant information (mainly organizational).
Documenting the Decision	*Knowledge Keeper (KK)* stores and documents the produced information and the decisions taken in the process for their future use in forthcoming selection processes.

2.2 Existing Resources for Supporting OTS Selection

From the state-of-the-art review, we found that researchers and practitioners have been dealing with COTS components selection for a quite time and several selection methods and tools have been proposed, for instance: CARE, OTSO, PECA, PORE, etc., see [9] for a survey. Moreover, in the last years the use of OSS components has brought out extraordinary research interest and specific selecting approaches have been put forward [2],[10],[11]. However, almost all of these proposals focus their efforts on the evaluation of OTS, instead of locating components in a huge and changing marketplace.

Component location is usually supported by component repository systems. However, although reusable component repositories have been an active research area for more than a decade, they have not yet received wide success in practice, mainly because of too heavy upstart and maintainance cost; and undercritical information relevance later on [12],[13]. Moreover, a similar challenge to deal with OTS repositories have also been recognized [14].

To further investigate the problems that small and medium companies face when selecting OTS, we have recently performed an explorative survey in some Norwegian companies [15]. Our results draw that the components identification complexity in industrial settings is actually twofold:

- *How to know which kind of components are available and which of them could be useful to solve a specific problem?* There is an increasing need for organizing the OTS information available to achieve more efficient and reliable selection processes [16]. However, the effort is considerable due to the size and variability of the software market and the difficulty to collect and update information. It is thus a hard task for enterprises, particularly for small and medium ones, which can not invest enough time, money, and effort into component management to gain qualified information.

- *How to find and process the information referred to those components to perform an effective evaluation?* Even when COTS and OSS components information are supported by their specific vendors and open source community projects respectively, the kind of information they contain is often not detailed enough, and usually unstructured, presented in many different forms (e.g., forums, documents, etc.), very difficult to be processed for an objective evaluation [6],[7],[17].

Furthermore, from the answers of our respondents we figure out that the *World Wide Web* is the most used means to find candidate components (i.e., search on available catalogues or specialized search engines) followed by colleague recommendations. We also asked about the resources they usually use to locate components and information about them, as well as the perceived utility of such information for performing the different OTS selection activities.

Summarizing the answers, in Table 2, we provide an excerpt of the most mentioned resources, their key characteristics as well as the utility of their information to the roles tasks.

Table 2. Some Available Web Resources for Supporting OTS Selection

Name	**Scope**	**Components Information**	**Support to the Roles**			
			MW	QE	S	KK
COTS Vendors	COTS	Non-Structured (**NS**)	*	*	*	-
OSS Project	OSS	**NS**	*	*	*	-
SourceForge.net	OSS	**NS**	√	-	-	-
ComponentSource.com	Mainly COTS	**NS**	√	-	-	-
Tigris.org	OSS-Soft. Engineering related	**NS**	√	*	*	-
OpenCores.org	OSS-IP	**NS**	√	*	-	-
KnowledgeStorm.com	Mainly COTS	**NS**	√	*	-	-
CMSmatrix.org	OTS-CMS	Semi-Structured (SS)	*	*	-	-
Messangingmatrix.com	OTS-Messaging	SS	*	*	-	-
TheServerSide.com	Java	**NS**	√	-	-	-
Freshmeat.net	Mainly OSS	SS	*	*	-	-
Forrester.com	Broad IT Solutions	**NS**	*	-	-	-
Gartner.com	Broad ITSolutions	**NS**	*	-	-	-

(√) supports the task (*) deals with some issues (-) does not deal with the task

In Table 3, we sum up our assessment of role-related current practices, their problems and implied challenges. The challenges can be summarized as the need of combining: Understandable Taxonomies, a Common Component Description Metamodel embracing all the informational dimensions for evaluating OTS, and a Reuse Infrastructure Support also feasible to small and medium organizations that can not invest enough time and money to manage it. We realize that though many efforts have been paid to deal with some of these challenges (e.g., the different web resources cited in Table 2, and methods and tools mentioned at the beginning of this section) there is no consensus of their utility. Therefore, there is a gap between such efforts and their realistic application [3],[6],[7],[18],[19],[20]. This drawback generates a barrier on adoption of OTS components in large industrial projects, since they make the selection process highly risky and expensive when applying complex evaluation criteria.

Table 3. Assessment of the role-related challenges for supporting OTS Selection

Role	Current Practice	Problem	Challenge
MW	• Proliferation of cataloguing initiatives from profit and non-profit organizations. • Catalogues containing brief and unstructured descriptions of some inventoried components. • Most catalogues do not have a clear rationale behind.	× Understanding and using the categorizations may be difficult. × Several descriptions of the same component.	✓ Understandable Taxonomies [18] ✓ Common Component Description Metamodel [6]
QE	• OTS providers do not provide structured and enough information for supporting evaluation and product quality assessment.	× Complex discovering and structuring of critical information.	✓ Component Description Metamodel embracing quality characterristics [17]
S	• Non-Technical information about the component is even more difficult to be located.	× Hard requirements negotiation. × Complex decision-making process.	✓ Component Description Metamodel embracing non-technical factors [19]
KK	• No support for organizations (mainly small & medium) that continuously select OTS to reuse their knowledge about them.	× Reuse of knowledge is usually tacit, leading to be lost if people are replaced.	✓ Reuse Infrastructure Support [20]

3. Proposed Approach

To deal with the mentioned challenges as a whole, we propose the GOThIC (Goal-Oriented Taxonomy and reuse Infrastructure Construction) method [20]. It relies on several industrial experiences undertaken under action-research premises and grounded theory. The method is intended to guide the construction of an OTS reuse infrastructure (repository) that provides well-founded and understandable taxonomies to organize all information related to them. OTS information is structured in a Component Description Metamodel (CDM) based on the ISO/IEC 9126-1 quality standard. This has been extended to support all the informational dimensions for selecting OTS and reusing the information about them [14]. In addition, some research tools have been developed for supporting the method, i.e., the DesCOTS system (Description, evaluation, and selection of COTS components) [21].

From the industrial evaluation of GOThIC we found some concerns regarding its practical use: heavy upstart cost (i.e., small and medium enterprises will not be able in the general case to adopt it), and difficulty to maintain complete and up-to-date information due to the highly changing nature of the OTS components marketplace. To overcome such issues, we propose to combine the GOThIC method with the creative and productive potential of "open-source collaboration". In this way, the OTS technology users (i.e., individuals, organizations, academic researchers, industrials) can be harnessed to work as a community dedicated to incrementally build and maintain an open OTS information repository. This will ensure smooth start-up and maintenance cost, as well as highly reliable information. Details of this strategy are described in next section.

4. Our Solution: An Open Wiki-Based portal for Sharing and Reusing OTS information

We use the potential offered by a Wiki-based portal to put forward our strategy. A Wiki (from the Hawaiian Wikiwiki meaning "fast") is a collaboratively created and iteratively improved set of web pages [22]. It is considered a powerful knowledge management tool that enables the creation of an incrementally growing system containing the shared knowledge of multiple sources in a centralized infrastructure/repository (i.e. a database server, an application server that runs the Wiki software, and a web server that serves the pages and facilitates the web-based interaction). Thus, exploiting some particular Wiki characteristics (based on the principles described by Wagner [22]) we have designed an *OTS-Wiki* portal. Its main high-level goals are summarized below:

- **Fostering an OTS Community and Incremental Population of Content.** The OTS-Wiki provides the web-based infrastructure for enabling OTS technology users to collaborate as a community in an open-source-like environment, see Fig. 1. Thus, *OTS Community* users are able, and even encouraged to share knowledge (e.g., experiences, components information, and vendor comments). Therefore, the incremental population of content in the portal based on the OTS Community participation is expected. We have designed proper templates and guidelines for editing and use in order to share the information in a structured way (as demonstrated in the Wikipedia, an on-line encyclopedia implemented as a Wiki).
- **Federating Actual Efforts for Locating and Selecting OTS Components.** In this collaborative environment, *OTS Community* users are encouraged to add (as hyperlinks) and comment the helpfulness of existing web-resources for locating OTS components (as those cited in Table 2, called OTS Web-Resources in Fig. 1). This is a way of having an up-to date federated list of actual web-resources that the *OTS Community* users can exploit. Besides the obvious advantage of using hyperlinks for allowing users to make connections and to drill down into detailed knowledge, hyperlinks are also a potential quality assurance mechanism and relevance indicator. Pages with many links to them indicate a highly useful page. This factor fosters the OTS-Wiki portal to act as a meta-portal for

promoting the progressive homogenization (complying with CDM) of the information contained in different OTS web resources. This is because such resources have an interest of being perceived as highly useful by the OTS Community users.

- **Enabling Systematic Support for Selecting and Evaluating OTS Components.** Having structured OTS information enables systematic support for evaluating and choosing components. We are integrating the DesCOTS system into the *OTS-Wiki* [21], as stated in Fig.1. It includes a set of tools that interoperate to support the whole OTS selection process: the *Quality Model Tool* allows defining quality models; the *OTS Evaluation Tool* allows evaluating components; the *OTS Selection Tool* allows defining requirements that drive the OTS component selection; and the *Taxonomy Tool* allows organizing OTS domains as a taxonomy supporting reuse of quality models. Nevertheless, some other existing or new tools can be developed or designed for using the structured OTS component information from the *OTS-Wiki* portal.

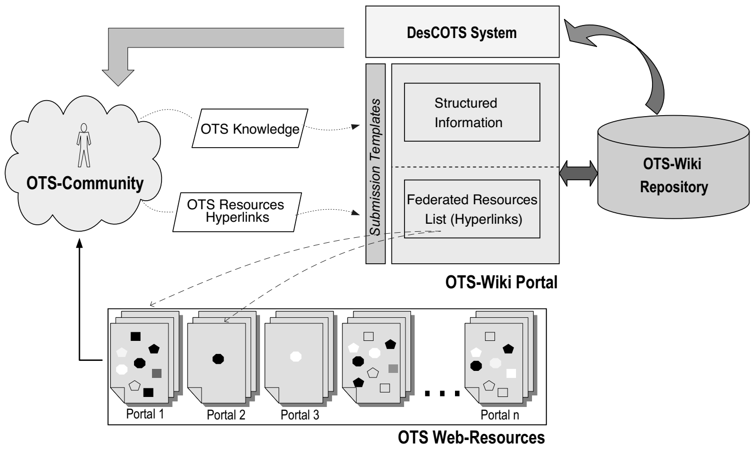


Fig. 1 OTS-Wiki Portal Main Interactions

In this scenario, any OTS Community user can use the OTS-Wiki portal as a meta-portal for providing support to: a) Locating OTS and information about them; b) Recording component information in a structured way; c) Maintaining and reusing such information; d) Getting tool support for performing selection processes. The structure of the information in the repository showed in Fig. 1 is detailed in next subsection.

4.1 Conceptual Model of the OTS-Wiki Portal Repository

Following the GOThIC method approach, the OTS-Wiki portal information is arranged as a goal-oriented taxonomy (composed of Market Segments and Categories) into the OTS-Wiki repository, as shown in Fig. 2. Taxonomy nodes have a generic CDM. The CDM is used as a template that can be instantiated with component information. For simplification purposes, we are not distinguishing at the moment versions of components; two different versions are treated as two different products. For each component, the DesCOTS system provides systematic support for OTS selection. This structure allows browsing the taxonomy and finding specific information [20].

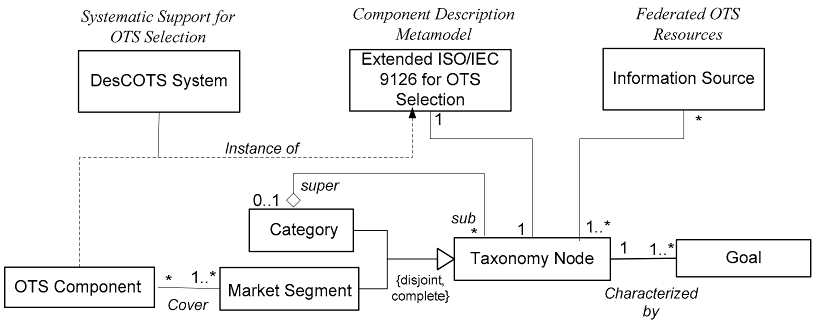
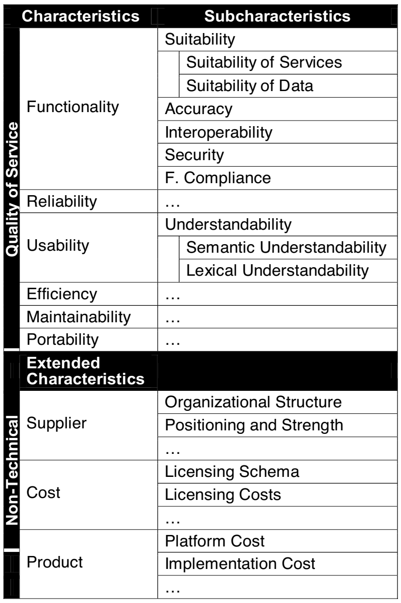


Fig. 2 Conceptual model for OTS-Wiki Portal Repository

4.1.1 Component Description Metamodel

The *Component Description Metamodel* (CDM) is based on the ISO-IEC 9126-1 software quality standard, and extended for covering all the informational dimensions needed for evaluating OTS components (functionality, quality of service, interoperability, non-technical factors and concepts definition) [14]. This structure has demonstrated to be useful for reusing information and helping the elicitation and (re)negotiation of requirements, making easier the identification of mismatches among components characteristics and the requirements in specific OTS selection processes [19]. Fig. 3 shows an excerpt of the structure of the *CDM*.



	Characteristics	Subcharacteristics	
Quality of Service	Functionality	Suitability	
			Suitability of Services
			Suitability of Data
		Accuracy	
		Interoperability	
		Security	
		F. Compliance	
	Reliability	…	
	Usability	Understandability	
			Semantic Understandability
			Lexical Understandability
	Efficiency	…	
	Maintainability	…	
	Portability	…	
	Extended Characteristics		
Non-Technical	Supplier	Organizational Structure	
		Positioning and Strength	
		…	
	Cost	Licensing Schema	
		Licensing Costs	
		…	
	Product	Platform Cost	
		Implementation Cost	
		…	

Fig. 3 Excerpt of the Component Description Metamodel

5. OTS-Wiki Portal Functionality Overview

In this section, we will provide some goal-based scenario excerpts of the OTS-Wiki prototype we have implemented in order to make explicit diverse mechanisms for reaching the high-level goals stated in section 4.

Fig. 4 associates the main scenarios to reach the OTS-Wiki portal high-level goals. From scenario 4a we realize that the OTS-Wiki portal has been designed as open and freely accessible in order to enable the OTS Community in an open source-like environment. Scenarios 4b, 4c and 4d show the refinement of the high-level goals explained in section 4 into other specific sub-goals or functionalities.

Goal:	**Fostering OTS Community**
Description	OTS Technology users are encouraged to work as a high performance team for reusing and sharing OTS Components Information in an Open and Freely accessible OTS-Wiki Portal.
Related goal(s)	1.-Incremental Population of Content 2.-Federation of OTS Resources 3.-Enabled Systematic Support for OTS Selection …
PostCondition(s)	Progressive Foundation of OTS Community

4a)

Goal:	**Incremental Population of Content**
Description	Users are encouraged to publish and share content they considered helpful to the OTS Community.
Related goal(s)	1.-Submit OTS Component Information 2.-Enabled Active Communication 3.-New Functionality Requested to the Community 4.-Enabled a Glossary Construction 5.-User Profiles to Personalize the Information …
PostCondition(s)	Incremental growth of the OTS-Wiki portal content

4b)

Goal:	**Federation of OTS Resources in OTS-Wiki**
Description	Users are encouraged to publish content that they consider may be helpful to the Community.
Related goal(s)	1. - User Introduces a new OTS-Web Resource Hyperlink to the Federated OTS Resources List by means of a template. 1.1. - System Records the hyperlink in the OTS-Wiki Repository.
	2. – User Introduces a File 2.1. – Resource is uploaded to the OTS-Wiki Respository.
	3. - User Provides a non-web reference 3.1. - Reference is Recorded in the OTS-Wiki Repository
PostCondition(s)	Incremental growth of the federated resources.

4c)

Goal:	**Enabled Systematic Support for Selection Process**
Description	Tools are provided to support the OTS selection activities automatically, using the standardized data from the repository. It is actually based on the DesCOTS functionality.
Related goal(s)	1.- User Requests automatic support for stating requirements 2.- User Requests automatic support for matching requirements with components. …
PostCondition(s)	User is Supported to perform and document his or her selection process. System Learns from each selection case (i.e. non-chosen components are recorded for being shown –by analogy- to later searches)

4d)

Fig. 4 Goal-based Scenarios designed to reach the OTS-Wiki High-Level goals

Fig. 5 shows some of the specific scenarios sub-goals:

Goal:	**Enabled Active Communication**
Description	Users are encouraged to maintain an active and fruitful communication among them.
Related goal(s)	1-User Creates a Discussion Board 2-User Participates in an existing Discussion Board 3-User Creates a chatting discussion 4-User Participates in an existing Chat …
PostCondition(s)	Incremental growth of information from the active communication.

5a)

Goal:	**Enabled Assisted Search**
Description	Users are provided with searching facilities to locate OTS components information.
Related goal(s)	1. - Searching similar terms in the OTS-Wiki Repository. 1.1. - Searching by Keywords 1.2. - Searching by Browsing
	2.- Showing Federated *OTS Resources List* were to find OTS component information
PostCondition(s)	The system shows all the related information found (e.g. actual users of the component, lessons learned, FAQs, forums, related experiences, integration cost, vendor helpfulness). Let non-found components serve as requirements for future/non-registered components.

5b)

Goal:	**New Functionality Requested to the Community**
Description	Users are provided with a Requesting Board area for requesting information of components functionality that do not already exist in the OTS-Wiki portal (but maybe in other portals) or new component functionalities to the Community.
Related goal(s)	1. - User Makes a Functionality Request 2. - User Answers a Functionality Request 3. - System generates a Request (from scenario 5b)
PostCondition(s)	The system manages the status of the requests.

5c)

Goal:	**Enabled A Glossary Construction**
Description	Users are encouraged to detail the meaning of unknown or confusing terms.
Related goal(s)	1. - User Adds a term to the Glossary 2. - User Associates terms related
PostCondition(s)	Incremental growth of the Glossary.

5d)

Fig. 5 Scenario excerpts to enable functionalities addressed to reach High-level OTS-Wiki Portal goals

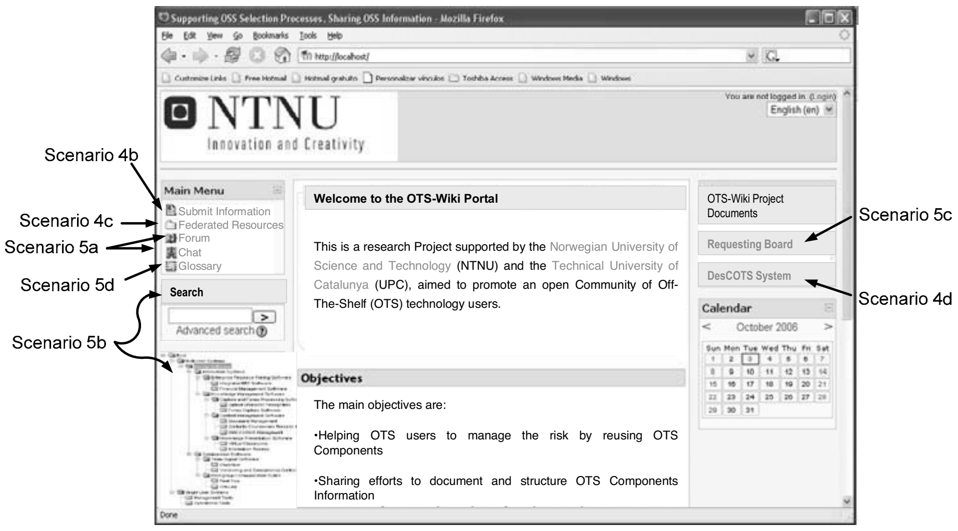
- Fig. 5a. *Enabled Active Communication*: diverse mechanisms (e.g. discussion boards, chat, distribution list, etc.) are provided to enable active communication among community users.

- Fig 5b. *Enabled Assisted Search*: searching in the OTS-Wiki portal may be performed by keyword or by taxonomy navigation. The taxonomy navigation we propose (already implemented in the DesCOTS system [21]) helps users to analyze their OTS selection problem and finding their suitable market segment by navigating through a hierarchical search tree, ruling out irrelevant nodes through a question-and-answer dialog. If the information requested does not already exist in the OTS-Wiki repository, the system shows the *Federated OTS Resources List* providing hyperlinks to different resources where the information could be found; and generates a *Functionality Request* (Scenario 5c).

- Fig. 5c. *New Functionality Requested to the Community*: users are able to request and discuss component functionality not found in the OTS-Wiki portal, or with no actual implementation (those for which information was found neither in the OTS-Wiki nor in any other portal). This could result in a competition among OSS communities and COTS providers to make such components, or even encouraging the creation of new OSS communities for supporting such functionality.

- Fig. 5d *Enabled Glossary Construction*: detailing the meaning of unknown or confusing terms is important because it is common in the OTS context that the same concept may be denoted by different names in different products or even worse, the same term may denote different concepts in different products. Therefore, main concepts should be clarified via explanation pages that comprise a Glossary. This glossary also serves to provide semantic relationships among concepts via hyperlinks.

Finally, in Fig. 6 we provide a snapshot of the actual OTS-Wiki prototype and relate its functionality with the scenarios described above. Some others functionalities to be incorporated are: to provide user profiles to personalize the information to the different roles needs, and case-base reasoning support for improving the searching processes and selection of multiple components.



Welcome to the OTS-Wiki Portal

This is a research Project supported by the Norwegian University of Science and Technology (NTNU) and the Technical University of Catalunya (UPC), aimed to promote an open Community of Off-The-Shelf (OTS) technology users.

Objectives

The main objectives are:

•Helping OTS users to manage the risk by reusing OTS Components

•Sharing efforts to document and structure OTS Components Information

Fig. 6 OTS-Wiki Portal functionalities related with described Scenarios

6. Ongoing Work

So far, we have an OTS-Wiki portal prototype combining Web-portal and Wiki technologies. It is expected to be fully operational before May 2007. Our next short-term goal is to populate the OTS-wiki repository with some broadly used OTS components as a way to give momentum to our approach. Thus, we are taking as a base the semi-structured components information from CMSmatrix (cited in Table 2), that comprises components related with the Content Management System market segment, in order to be transformed into our proposed CDM.

On the other hand, some informational quality concerns have being discussed as user control and information ownership to provide high-quality information.

7. Future Work

Our intended main goal is to empirically study how the GOThIC method addresses the different issues related to OTS component selection and evaluation, as well as the effects and results of this kind of open-source-like collaboration concept for dealing with OTS selection challenges, and the problems reported with the use of repositories [12],[13]. Some metrics intended to be used are: support perceived by users, ability to enable the OTS Community, ability to promote homogenization, promotion of OSS communities, information reuse, etc. Moreover, given the social computing nature of our proposal, its functionality is going to be incrementally improved depending on the OTS Community trends and needs.

8. Conclusion

The proposal presented here is a feasible and incremental way of dealing with the drawbacks of OTS selection processes as well as the problems reported with the use of repositories. It is done combining the GOThIC method and the "open-source collaboration" approach in a social computing environment:

- It represents a feasible support to improve OTS selection, integration and maintenance processes as well as knowledge reuse, mainly in small and medium companies that are not able to invest enough money and time to manage a repository themselves (smooth start up and maintenance cost).
- The combination of ease and speed of publishing contents, together with the ability of engaging the potential *OTS Community* into the structured knowledge creation process, enables *OTS-Wiki* to become a quality platform for very large and up-to-date OTS knowledge repositories that acts as a Meta-portal for structuring the OTS unstructured information contained in other portals (this is best illustrated by the Wikipedia).
- It allows the incremental growth of a component base, where each component is linked to a community of interested users.

- It fosters the (re)use of OTS components and promotes communities to address requirements with no actual implementation.
- The *OTS Community* interaction may address not only the challenges mentioned in section 3, but also the actual research efforts into the real needs and trends of the *OTS Community*.

9. Acknowledgements

This work has been partially supported by the Spanish MEC TIN2004-07461-C02-01 project and the Norwegian COSI (Co-development and using inner & Open source in Software Intensive products) project. C. Ayala´s work has been partially supported by the Mexican Council of Science and Technology (CONACyT) and the Agència de Gestió d'Ajuts Universitaris i de Recerca (European Social Fund).

10. References

1. Brownsword, L., Oberndorf, T., Sledge C.A. "Developing New Processes for COTS-Based Systems" IEEE Software, Vol. 17, No. 4; July-August 2000. pp.48-55.
2. Madanmohan, T.R., De, R. "Open Source Reuse in Commercial Firms" *IEEE Software* 21(6). 62-69.
3. Ankolekar, A., Herbsleb, J., Sycara, K. "Addressing Challenges to Open Source Collaboration with Semantic Web". In proceedings of 3rd Workshop on Open Source Software Engineering, the 25th International Conference on Software Engineering (ICSE). 2003. Portland, Oregon, USA, pp 9-14.
4. Torchiano, M., Morisio, M. Overlooked Aspects of COTS-Based Development. *IEEE Software*, March/April 2004, pp 88-93.
5. Vitharana, P., Zahedi, F., Jain, H. "Knowledge-Based Repository Scheme for Storing and Retrieving Business Components: A Theoretical Design and Empirical Analysis". *IEEE Transactions on Software Engineering*. Vol. 29(7), 2003, pp 649-664.
6. Réquilé-Romanczuk, A., Cechich, A., Dourgnon-Hanoune, A., Mielnik, J.C., "Towards a Knowledge-based Framework for COTS components Identification" ICSE-MPEC05, ACM Press, 2005; pp 1-4.
7. Simmons, G.L., Dillon, T.S. Towards an Ontology for Open Source Software Development. In IFIP International Federation for Information Processing, Volume 203, Open Source Systems, eds. Damiani, E., Fitzgeralg, B., Scacchi, W., Scotto, M., Succi, G., (Boston:Springer), pp 65-75.
8. Li, J. Process improvement and risk management in Off-the-Shelf Component-based development. PhD Thesis 2006. Norwegian University of Science and Technology (NTNU). ISBN 82-471-7920-2. 289. http://www.idi.ntnu.no/grupper/su/publ/phd/li-phdthesis-22jun06.pdf
9. Ruhe, G. "Intelligent Support for Selection of COTS Products" Proceedings Web Databases and Web Services 2002. LNCS 2593, pp. 34-45.
10. Wheeler, D.A.: How to Evaluate Open Source Software / Free Software (OSS/FS) programs. URL http://www.dwheeler.com/oss_fs_eval.html.
11. van der Berg, K. "Finding Open Options". Master Degree Thesis. Tilburg University. 2005.
12. Morisio, M., Ezran, M., and C. Tully, "Success and Failure Factors in Software Reuse," *IEEE Trans. Software Eng.*, vol. 28, no. 4, 2002. pp. 340-357.

13. Poulin, J. S. Populating Software Repositories: Incentives and Domain Specific Software. *J Systems and Software* 1995. Elsevier, pp 187-199.
14. Ayala, C., Franch, X. "Domain Analysis for Supporting Commercial Off-The-Shelf Components Selection" 25th International Conference on Conceptual Modeling (ER 2006). Tucson, Arizona, USA. November 2006. LNCS 4215, pp 354-370.
15. Gerea, M. "Selection and Evaluation of Open Source Components" Department of Computer and Information Science. Norwegian University of Science and Technology (NTNU). http://www.idi.ntnu.no/grupper/su/fordypningsprosjekt-2006/gerea-fordyp06.pdf
16. Ayala, C., Franch, X. "Transforming Software Package Classification Hierarchies into Goal-Based Taxonomies". In Proceedings of the 16th Database and Expert Systems Applications Conference (DEXA), LNCS 3588, 2005. pp 665-675.
17. Bertoa, M.F., Troya, J.M., Vallecillo, A. "A Survey on the Quality Information Provided by Software Component Vendors". In Proceedings of the 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE), 2003, pp 25-30.
18. Cechich, A., Réquilé-Romanczuk, A., Aguirre, J., Luzuriaga, J.M. "Trends on COTS Component Identification and Retrieval" In Proceedings of 5th International Conference on COTS-Based Software Systems (ICCBSS), IEEE Computer Society, 2006, pp 90-99.
19. Carvallo, J.P., Franch, X. "Extending the ISO/IEC 9126-1 Quality Model with Non-Technical Factors for COTS Components Selection" In Proceedings on the Workshop on Software Quality (WOSQ'06). IEEE Computer Society. 2006, pp. 9-14
20. Ayala, C., Franch, X. "A Goal-Oriented Strategy for Supporting Commercial Off-The-Shelf Components Selection" 9th International Conference on Software Reuse (ICSR). June 2006. LNCS 4039, pp 13-24.
21. Grau, G., Carvallo, J.P., Franch, X., Quer, C., "DesCOTS: A Software System for Selecting COTS Components" In Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04). IEEE Computer Society, pp 118-126.
22. Wagner, C. "Wiki: A Technology for Conversational Knowledge Management and Group Collaboration". Communications of the Association for Information Systems Vol.13, 2004. 265-289.

From the Cathedral to the Bazaar: An Empirical Study of the Lifecycle of Volunteer Community Projects

Andrea Capiluppi[1], and Martin Michlmayr[2]
1 University of Lincoln, UK, acapiluppi@lincoln.ac.uk
2 University of Cambridge, UK, martin@michlmayr.org

Abstract. Some free software and open source projects have been extremely successful in the past. The success of a project is often related to the number of developers it can attract: a larger community of developers (the `bazaar') identifies and corrects more software defects and adds more features via a peer-review process. In this paper two free software projects (Wine and Arla) are empirically explored in order to characterize their software lifecycle, development processes and communities. Both the projects show a phase where the number of active developers and the actual work performed on the system is constant, or does not grow: we argued that this phase corresponds to the one termed 'cathedral' in the literature. One of the two projects (Wine) shows also a second phase: a sudden growing amount of developers corresponds to a similar growing output produced: we termed this as the `bazaar' phase, and we also argued that this phase was not achieved for the other system. A further analysis revealed that the transition between `cathedral' and `bazaar' was a phase by itself in Wine, achieved by creating a growing amount of new modules, which attracted new developers.

1 Introduction

Prominent free software (or open source software, OSS) projects such as Linux [32], Apache [27] and FreeBSD [18] have been extremely successful. Anecdotal evidence has been used in the past to characterize successful OSS projects: users/developers acting as "more eyeballs" in the correction of bugs, developers implementing new features independently, skillful project managers dealing with a mostly flat organization, and the resulting coordination costs [28].

Previous studies have provided empirical evidence on the process of successful OSS projects: the definition of various types of developers has been discussed for the Mozilla and the Apache projects, justifying different levels of effort [27], and claiming that the first type (core developers) contribute to the success of a system.

Please use the following format when citing this chapter:

Capiluppi, A. and Michlmayr, M., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 31–44.

Also, social network analyses have shown communication and coordination costs in successful OSS projects [21].

In all these cases, successful projects are studied and characterized, but an analysis in their earlier inception is not given. Therefore, empirical studies on whether the project always benefited of a large number of developers, or built instead its bazaar through several years, are still missing. In order to tackle this missing link, this paper explores the evolution and development processes of two OSS systems, the Wine (a free implementation of Windows on Unix) project and the Arla file system. The first system has been widely adopted and developed by many developers. Arla, on the other hand, is still in a `cathedral' phase when compared Wine: fewer developers are currently collaborating towards its development.

The aim of this paper is to empirically detect and characterize the phases achieved by these two systems, to illustrate whether one phase consequently follow the other, and to establish one of these phases as a `success' for an OSS project. If this is the case, sharing the empirical guidelines on how to achieve this transition could help developers to work on the benefits of the bazaar phase.

Structure of the paper: in Section 2, a theoretical background will be given, as well as two research questions, based on OSS communities. Also, a description of the approach used to acquire and analyses the data employed will be presented. The data will be used to test the presented questions. Section 3 will describe the phases observed in the two systems from the point of view of the activities of developers. This section will also give a detailed description of the activities that underpin the success of a OSS system, as observed in the proposed case studies. Section 4 will deal with related work in this (and other) areas, identifying the main contributions of this paper, and will discuss a number of questions raised in this paper that need further empirical exploration. Finally, Section 5 will give conclusions on the overall process and lifecycle of OSS systems, as well as possible future research directions.

2 Background Research

One of the authors, in a previous work [29], presented a theoretical framework for the activities and phases of the lifecycle of OSS projects. The objective was to provide a more systematic approach for the development of OSS projects, to increase the likelihood of success in new projects. In this paper, the objective is to empirically evaluate the theory contained in that work through two case studies, and to report on best practices of actually successful OSS projects. Since previous studies have shown that many OSS projects must be considered failures [3, 7], it is argued that the latter ones lack some of the characteristics as described in [29], notably the transition between the closed (or `cathedral') and the open (or `bazaar') styles. In his popular essay “The Cathedral and the Bazaar", Eric S. Raymond [28] investigates development structures in OSS projects in light of the success of Linux. The terminology of the `cathedral' and the `bazaar' introduces both a closed approach, found in most commercial entities, where decisions on large software projects are taken by a central management; and an open one, where an entire community is in charge of the whole system.

Instead of viewing these approaches as diametrically opposed, as originally proposed by Raymond, this paper considers these as complimentary events within the same OSS software project. Figure 1 illustrates three basic phases, which this research argues a successful OSS project undergoes. The initial phase of a OSS project does not operate in the context of a community of volunteers. All the characteristics of cathedral style development (like requirements gath- ering, design, implementation and testing) are present, and they are carried out in the typical style of building a cathedral, that is, the work is done by an individual or a small team working in isolation from the community [5]. This development process shows tight control and planning from the central project author, and is referred to as `closed prototyping' by Johnson [17].

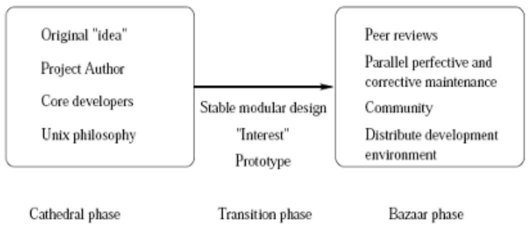


Fig. 1. *OSS development lifecycle*

In order to become a high quality and useful product, [29] argued that an OSS project has to make a transition from the cathedral phase to the bazaar phase (as depicted by the arrow in Figure 1). In this phase, users and developers continuously join the project writing code, submitting patches and correcting bugs. This transition is associated with many complications: it is argued that the majority of free software projects never leave the cathedral phase and therefore do not access the vast resources of manpower and skills the free software community offers [7].

2.1 Research questions

In this paper, historical data on code modifications and additions of large (subsystems) or small scale (modules) sections of a software system are analyzed in order to track how the studied systems evolved over time. Two research questions are presented here: the historical data will be then tested against them, and the results will be evaluated in the next section. The first is based on out- put obtained from input provided, the second on what new developers tend to work on when joining a OSS project. The research questions can be formulated as follows (metrics used to asses each question are also provided):

i) research question 1: the `bazaar' phase involves a growing amount of developers, who join in a self-sustaining cycle. The output obtained in a bazaar phase follows a similar growing trend. OSS projects, while still in the `cathedral' phase, do not benefit from a growing trend in input provided and output achieved.

ii) research question 2: new developers, when joining a software project, tend to work on newest modules first, either by creating the modules themselves, or by contributing to a new module. This can be rationalized saying that new developers might not need insights on all the preexisting functionalities of a system, thus preferring to develop something new. This research question will be used to gather further insights on how Wine could achieve a bazaar phase.

2.2 Empirical approach

The empirical approach involves the extraction of all changes embedded in sources of information of both input (effort provided by developers) and output (that is, additions or changes of subsystems and modules). In the following analysis, the ChangeLog file, recording the whole change history of a project, has been used rather than an analysis of the projects' CVS repositories. From previous research it is known [10, 22] that different development practices have an influence on the best data source, and the ChangeLog file offers more reliable information in the selected case projects [6, 12, 30].

The steps to produce the final data can be summarized in: parse of raw data, and extraction of metrics. As part of the first step, automated Perl scripts are written to parse the raw data contained in the ChangeLog and to extract pre-defined data fields. The data fields which will be considered in this study are: name of the system, name of the module, name of the subsystem containing that module, date of creation or change and unique ID (name and email) of the developer responsible for the change.

2.2.1 Raw data extraction

The analyzed ChangeLog files follow very regular an- notating patterns, thereby allowing a straightforward analysis of the history of changes in a project in a semi-automated way. The following steps have been performed during the extraction of the raw data:

1 – Identification of dates: it was observed in the studied cases that each touch was delimited by a date, using the following or a similar pattern: for example, YYYY-MM-DD, as in “2000-12-31”. Each touch can be associated with one or more than one developers; also, each touch can be associated with one or more than one modules. For each touch there is one and only one date.

2 – Affected modules and subsystems: each touch affects at least one *file*, and is recorded with a plain-text description. In some cases the same touch affects many files: these modifications are referred to the same date. Subsystems are extracted as the *folder* containing the affected file.

3 – Details of developers: All touches concern at least one developer, displayed in various forms inside of the description of the touch. If more than one developers are responsible for a touch, they are recorded together within the touch.

4 – Derivation of metrics: Counts were derived of both effort provided by developers and work produced creating new modules and amending existing ones.

2.2.2 Metrics choice and description

The analysis of the two OSS systems involved three types of metrics, used differently to discuss the research questions. A list is proposed in the following:

i - Input metrics: the effort of developers was evaluated by counting the number of unique (or distinct, in a SQL-like terminology) developers during a specific interval of time. The chosen granularity of time was based on months: different approaches may be used, as on a weekly or on a daily basis, but it is believed that the month represented a larger grained unit of time to gather the number of active developers. This metrics was used to evaluate the first research question. For instance, in February 2006 it was found that the Wine system had 73 distinct developers who wrote code for this system in that month.

ii - Output metrics: the work produced was evaluated by counting the touches to modules or subsystems during the same interval of time. Smaller- grained metrics, like lines of code, were not considered in this study: evaluating how many lines of code are produced by OSS developers could be subject to strong limitations[1]. In the following section this metric will be used also as an indicator of parallel development work performed in successful projects. This metrics was also used to evaluate the first research question. As above, in February 2006 it was detected that the Wine system had 820 distinct modules which were touched in that month.

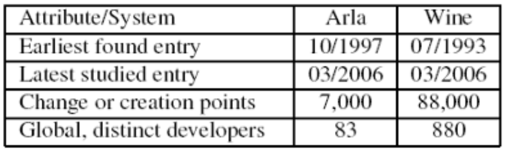
iii - New Input and Output metrics: the newly-added effort was evaluated counting the new developers joining the project. The work produced by these new developers was also isolated: the objective is to determine how much of this work has been focused on existing parts of the system, and how much goes to new parts. This metrics served to evaluate the second research question, i.e. to explore if new developers tend to work either on old or new parts of the system. As above, in February 2006 it was detected that the Wine system had 73 new developers (i.e. not detected in any of the previous touches). It was also empirically detected that these new developers worked in part on old modules, and in part on new modules, i.e. added in the same month. It was observed that 75% of their work concerned newer modules, and 25% on existing modules.

2.3 Case studies

The choice of the case studies was based on the recognized, objective success of one of the systems (Wine), while the second analyzed system (Arla) seems to have suffered from an inability of recruiting new developers, and achieved a much smaller overall size. Both of them have been used in the past for other empirical case studies, and their development style and growth pattern have been extensively studied.

The authors recognize that the two systems have two very different application domains: Wine is a tool to run Windows applications on Linux and other operating systems, while Arla is a networked file system. The main objective of the present study was not to evaluate the exogenous reasons behind successfully recruiting projects (like the presence of recognized "gurus" in a project, the good reputation of the existing community, etc [9]). On the contrary, this study focuses on evaluating the presence of three different stages in successful projects. The research presented here proposes a theoretical framework for OSS projects, independently from their domain, and empirically evaluates the mechanisms of forming a community around OSS projects.

The choice of the information sources was restricted to two classes of items, the CVS commits and the ChangeLog



Attribute/System	Arla	Wine
Earliest found entry	10/1997	07/1993
Latest studied entry	03/2006	03/2006
Change or creation points	7,000	88,000
Global, distinct developers	83	880

Table 1: *summary of information in the two systems.*

[1] Lines of code produced are biased by the skills of the developer, the programming language and, in general, the context of the modifications.

records. The CVS repository of Arla was found to be *incomplete*, since it does not contain the complete evolution history of the project. This is probably due to the fact that the CVS has been adopted at some point after the project's first inception. It was also observed that the CVS server of Wine is *inaccurate*: a query for active developers shows only 2 committers, against a much larger number of developers found in the ChangeLog records. That probably means a restriction in the write access to the Wine CVS. ChangeLogs were therefore preferred over CVS logs.

As a means to characterize the two systems, Table 1 displays some basic information about their ChangeLog files, the time span, and the amount of distinct developers which were found actively contributing to the project.

3 Results and discussion of the phases

In the following section, the two research questions are discussed, and the three phases (cathedral and bazaar, separated by a transition phase) as presented in [29] are evaluated, based on the empirical data from the case studies. Apart from this evaluation, it is also planned to identify some practical actions that OSS developers should consider in order to enhance the evolutionary success of their projects, and to ease the transition between the cathedral and the bazaar phases.

3.1 The cathedral phase

One of the main differences between closed, traditional software and OSS development is the ownership of the code. In the first environment, the development is typically driven by a team of individuals, while users do not contribute to, nor access the source code. In the latter, potentially everyone has the right to access and modify the source code underlying an application. It is argued that a typical OSS system will follow a cathedral approach in its first evolution history.

Arla system – input: Figure 2 (left) shows the distribution of distinct developers per month in the Arla system. Even though a sum of over 80 developers have contributed code, patches and fixes to the project (see Table 1), the number of distinct developers working on the development each month is much lower: on average only about five distinct developers work on the code base each month. As stated above, the first research question is not confirmed by the empirical findings: in the Arla project, the evolution of distinct, active developers in a month shows a regular, constant pattern.

Arla system – output: Figure 2 (right), on the contrary, shows the amount of distinct modules and subsystems that Arla developers have worked on since its inception: the distribution is fairly regular, and that could mean that new developers, when joining the project, are not expanding it into new areas, but that they rather work on existing functionality, together with the core developers. This will be tested in the section dedicated to the transition phase. These output findings, i.e. a constant, not growing pattern in output produced, confirm that the first research question does not apply for the Arla system.

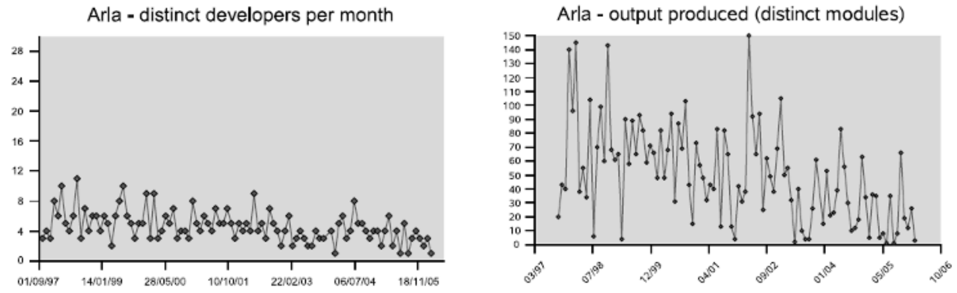


Figure 2: Development input (left) and output produced (right) in Arla

While these findings do not necessarily imply that Arla is a failure compared to Wine (as in the overall amount of developers from Table 1), it raises some interesting questions: for instance, it should be studied why only a small, but constant, number of developers is contributing code. As a possible explanation of its (reduced) success in recruiting new developers, one could argue that the system could be perceived as mature already [8], and that little further work was needed. Similar problems have been observed in the past for the OpenOffice.org and Mozilla systems: they represent two extremely complex applications and required a huge investment in the study, before developers could actually contribute directly.

In the next sections, practical guidelines will be evaluated on how an OSS system could tackle the issues faced by the Arla project, and in order to benefit of the efforts of a larger pool of developers.

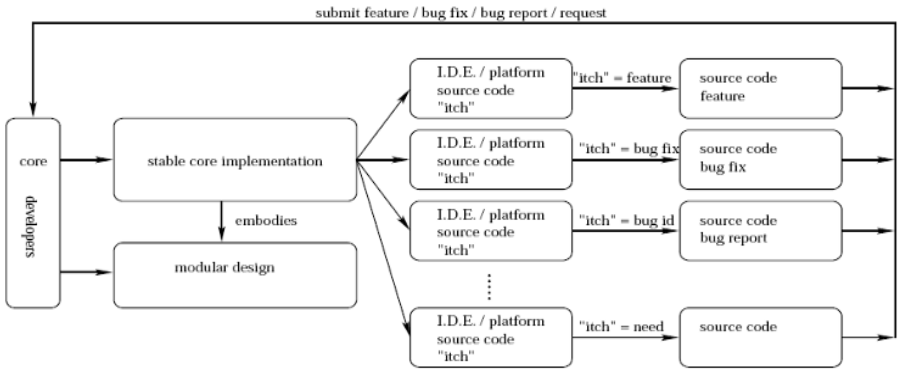


Figure 3: Detailed bazaar phase

3.2 Bazaar phase

The aim of many OSS projects is to reach a stage where a community of users can actively contribute to its further development. Some of the key characteristics of the bazaar phase are visualized in Figure 3, and can be summarized as follows:

- Contributions: the bazaar style makes source code publicly available and contributions are actively encouraged, particularly from people using the software. Contributions can come in many different forms and at any time. Non-technical users can suggest new requirements, write user documentation and tutorials, or point out usability problems (represented as low-level "itches" in Figure 3); technical users can implement features, fix defects and even extend the design of the software (the high-level "itches" of Figure 3).

- Software quality: increased levels of quality comes from thorough, parallel inspections of the software, carried out by a large community of users and developers. These benefits are consistent with software engineering principles: the `debugging process' of a OSS project is synonymous with the maintenance phase of a traditional software lifecycle.
- Community: a network of users and developers review and modify the code associated with a software system. The old adage "many hands make light work" is appropriate in describing the reasons for the success of some OSS projects [27].

Wine system – input: From the empirical standpoint, Figure 4 (left) shows the distribution of distinct developers per month in the Wine system. In total, over 800 developers have contributed code, patches and fixes (Table 1). Even though this project has a longer time span, which could have facilitated the growth of a developers basis, a clear distinction between a first phase (cathedral) and a later phase (bazaar) can be identified in the number of developers. Around July 1998, the Wine system has undergone a massive evolution in the number of distinct developers involved in the project. The sustainability of this new bazaar phase is demonstrated by the further, continual increasing number of new distinct developers in the Wine system. The first research question finds an empirical evidence analyzing the Wine system, a growing pattern of active developers signals the presence of the bazaar

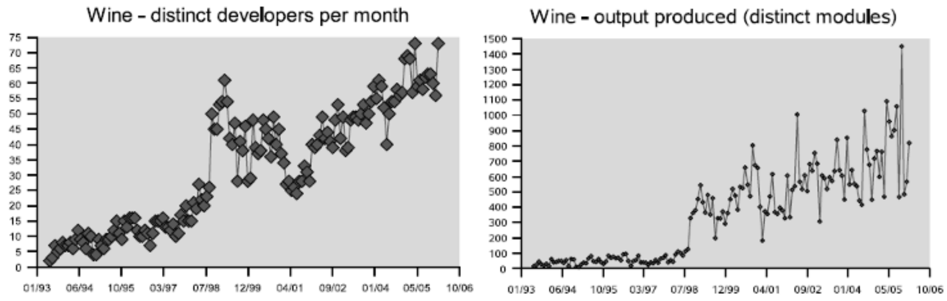


Figure 3: Development input (left) and output produced (right) in Wine

phase. The sustainability of the input process is visible in the ever-changing amount of distinct developers which participate in the evolution of the system.

Wine system – output: The bazaar phase is characterized by an open process in which input from volunteers defines the direction of the project, including the requirements. The initial implementation is mainly based on the requirements of the project author. In the bazaar phase, projects benefit from the involvement of a diverse range of users (with different requirements) who work together to increase the functionality and appeal of the software.

This parallel development behavior is achieved successfully in the Wine project. During the investigation of this system, the evolving scope of the project became apparent through the amount of distinct modules which developers work on each month. Figure 3 (right) shows the amount of distinct modules and subsystems that developers have worked on since its inception: the distribution is growing abruptly around the same time when an increase of distinct authors is observed Figure 3, right). This means that the project, with new developers joining constantly, is actively expanding it into new areas. The growing pattern of active developers sustains a growing pattern of output produced: as above, the first research question helps signaling the presence of the bazaar phase when such a growing pattern occurs.

3.3 Transition phase – defining new avenues of development

The theoretical framework represented in Figure 1 assigns a fundamental role to the transition phase, since it requires a drastic restructuring of the project, especially in the way the project is managed. One important aspect is commencing the transition at the right time. This is a crucial step and a hurdle many projects fail to overcome [11]. Since volunteers have to be attracted during the transition, the prototype needs to be functional but still in need of improvement [17, 28, 2].

If the prototype does not have sufficient functionality or stability, potential volunteers may not get involved. On the other hand, if the prototype is too advanced, new volunteers have little incentive to join the project because the code base is complex or the features they require have already been implemented. In both cases, adding future directions to the system could provide potential new developers further avenues for the development.

Based on the second research question, new developers, when joining a software project, tend to work on new modules, rather than old ones. As a consequence, the core developers should expand the original system into new directions and provide new code to work on: this would foster the recruitment of new developers and facilitate the transition phase.

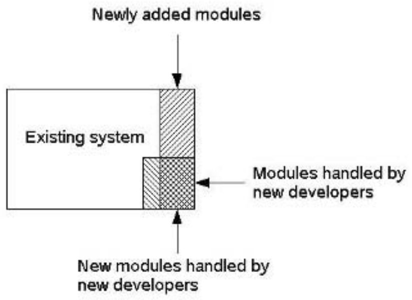

Newly added modules
Existing system
Modules handled by new developers
New modules handled by new developers

Figure 4: *Design of research question 2.*

To evaluate this question, an experiment was designed: at first, the newly added modules were extracted in every month. In parallel, the amount of new developers was also extracted. Finally, what new developers worked on was defined as the percentage of new modules they handled: Figure 4 graphically summaries this process.

The empirical results were extracted for the two systems Arla and Wine and are displayed in a box-plot, spanning all the releases for the two systems. Figure 8 is a description, on a percentile basis, of the modules as handled by newest developers.

Transition achieved – Wine: this system reveals that new developers, when joining the project, tend to work more easily on new modules than on older ones. In fact, more than 50% (on average) of what they work on is newly added in the same month, either by themselves or the core developers (right boxplot of Figure 5). Also, the average value of the boxplot was found to be larger when considering only the `bazaar' phase of Wine.

This first result is confirmed by plotting the amount of new modules created by the developers (Figure 5, right). A growing pattern is detected, similar to the one observed in the global evolution of the system (Figure 3): new developers join in, working on newest parts of the code, while core developers sustain the community of the project by continuously adding new modules.

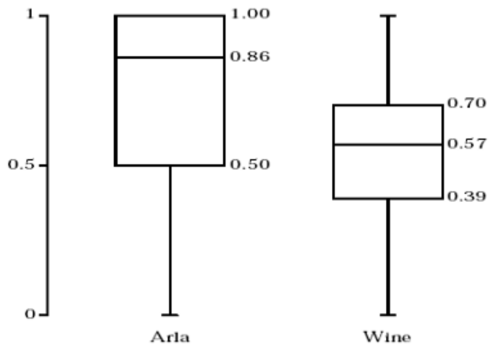


***Figure 5:** Description of effort for new developers*

Transition not achieved – Arla: this second system provides a much more interesting box-plot: the tendency of new developers is clearly towards working on something new, rather than on old modules (left boxplot of Figure 5). The main difference with the Wine project is that, for most of the periods, there are no new developers joining in the Arla development. Based on the assumptions of the second research question, new developers still prefer to start something new, or work on newly added code: still, this project could not ease the transition phase by not recruiting new developers. Therefore, it is possible to conclude that the original developers in Arla failed in providing new directions for the system, by creating new modules or subsystems. This conclusion is backed by the amount of new modules created by the developers (Figure 6, left): a decreasing pattern is detected, which confirms that new developers (and the community around the project), albeit willing to work on the system, were not adequately stimulated by the core developers.

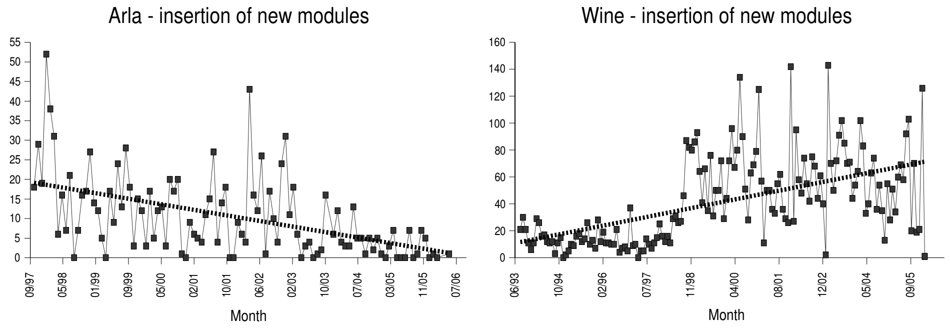


***Figure 6:** Creation of new modules in the Arla and Wine systems*

In summary, considering the second research question stated above, we found similar evidences for both the systems: when joining the development of an OSS system, new developers tend to work on (i.e., add or modify) new modules rather than old ones. As a proposed corollary to these results, the transition to a bazaar phase should be actively sought by the core developers: potential new developers should be actively fostered adding new ideas or directions to the project.

4 Related work

In this section the present work is related to various fields, specifically empirical studies on software systems and effort evaluation. Since this work is in a larger research context, related to the study of the evolution of OSS systems, empirical studies of OSS are also relevant to this research.

The earliest studies of the evolution of software systems were achieved through the proprietary operating system OS/360 [4]. The initial studied observed some 20 releases of OS/360, and the results that emerged from this investigation, and subsequent studies of other proprietary commercial software [20], included the SPE program classification and a set of laws of software evolution.

The present research has been conducted similarly, but evaluating both the input (as effort) provided, and the output (as changes made to the code base) achieved. The research questions which this paper is based upon derives from [29], and is based on the presence of two distinct phases in the software lifecycle of OSS systems, namely the cathedral phase and the bazaar phase [28]. This in contrast with Raymond's suggestion that the bazaar is the typical style of open source projects [15, 28]: an empirical evaluation was achieved by studying the lifecycle of two large free software projects, of which only one has made the transition to the bazaar phase and attracted a large community of developers. It is believed by the authors that too much emphasis has been put on highly popular projects in the past which are not necessarily representative of the OSS community as a whole [13, 15, 16, 26]. Few projects make a transition to the bazaar, attracting a large and active developer community along the way.

Having a large bazaar surrounding a project has several advantages, such as the ability to incorporate feedback from a diverse base of users and developers. Nevertheless, this is not to say that projects which are not in the bazaar phase are necessarily failures – they neither have to be unsuccessful nor of low quality.

Interestingly enough, in contrast to Raymond's model, there are a number of applications, such as GNU *coreutils* and *tar*, which form a core part of every Linux system and which clearly follow the cathedral. Similarly, there are many projects entirely developed by a single, extremely competent developer which show high levels of quality. Due to the lack of better theories and empirical research, quality in OSS projects is explained through the bazaar with its peer review [1, 26, 28]. However, not every project with high quality actually exhibits a large bazaar and significant peer review.

A project in the cathedral phase can be highly successful and of high quality [31]. However, there are some restrictions a project in the cathedral phase faces as well as a number of potential problems which are less severe if the project had a large developer community. For example, while it is possible for a single developer to write an application with a limited scope (such as a boot loader), only a full community can complete a project with a larger scope (such as a full desktop environment). Furthermore, a project written by one developer may be of high quality but it also faces a high risk of failure due to the reliance on one person who is a volunteer [23, 25]. Having a large community around a project makes the project more sustainable.

This discussion shows the lack of research in a number of areas related to OSS projects. While a uniformed model for all OSS projects has been assumed in the past, it is increasingly becoming clear that there is a great variety in terms of development processes [9, 19, 14]. Better theories about success and quality in OSS projects are needed [24], as are further comparisons between projects with different levels of success and quality. Finally, it should not be assumed that the bazaar is necessarily the optimal phase for every project, or that it is not associated with any problems. There is a general assumption that it is beneficial for a OSS project to be open, but too much openness can also be harmful when it leads to incompetent developers or people who demotivate important contributors getting involved [9].

5 Conclusions and future work

Successful OSS projects have been studied and characterized in the past, but an empirical demonstration on how they achieved their status has not been proven yet. In order to tackle this missing link, this paper has presented an empirical exploration of two OSS projects, Arla and Wine, to illustrate different phases in their lifecycle, their development processes and the communities which formed around them. Their ChangeLog records were analyzed and all the changes and additions, performed by the developers over the years, were recorded.

The assumption underpinning this paper is that the `cathedral' and `bazaar' phases, as initially proposed and depicted by Raymond in [28], are not mutually exclusive: OSS projects start out in the cathedral phase, and potentially move to a bazaar later. The cathedral phase is characterized by closed development performed by a small group or developer, with much in common with traditional software development. The bazaar phase exploits a larger number of volunteers who contribute to the development of the software through defect reports, additional requirements, bug fixes and features. The transition between the two phases was argued to be by itself a phase too, which has to be accommodated by specific, active actions of the core developers or project author. It was also argued that this transition is a necessary factor for truly successful and popular projects.

A first research question has proposed the study of the difference between the cathedral and the bazaar phases: the first system (Arla) has remained, through its lifecycle, an effort of a limited number of developers, or in a cathedral phase. It was also argued that this should not be interpreted as a sign of the overall failure of an OSS project, but as a potentially missed opportunity to establish a thriving community around a project. On the contrary, the second system (Wine) only shows an initial phase that is similar to what observed in the Arla system: a second, longer phase (bazaar) has a growing amount of active developers and a continuous expansion of the system.

Through a second research question, the focus was moved to the preferences of new developers joining an OSS project: results on both the systems show that new developers prefer to work on newly added modules, rather than older ones. In the Wine system, existing developers eased the transition phase by adding many new modules which new developers could work on. On the other hand, new developers in

Arla, although eager to work on new code, were not yet given enough new directions of the project, and an overall poor ability in recruiting new developers was resulting.

The future work has been identified in a replication of the study with other OSS projects, especially those belonging to the same application domain: the results as obtained in this study have analyzed the creation of a community from a neutral point of view, that is, without considering exogenous drivers. Our next step is to introduce these drivers into the research, and analyze large projects which currently compete with each other for the scarce resource of developers.

References

[1] A. Aoki, K. Hayashi, K. Kishida, K. Nakakoji, Y. Nishinaka, B. Reeves, A. Takashima, and Y. Yamamoto. A case study of the evolution of jun: an object-oriented open-source 3d multimedia library. In Proceedings of the 23rd International Conference on Software Engineering, pages 524-533, Toronto, Canada, 2001.

[2] B. Arief, C. Gacek, and T. Lawrie. Software architectures and open source software – where can research leverage the most? In Proceedings of the 1st Workshop on Open Source Software Engineering, Toronto, Canada, 2001.

[3] R. Austen and G. Stephen. Evaluating the quality and quantity of data on open source software projects. In Proceedings of 1st International Conference on Open Source Systems, Genova, Italy, June 2005.

[4] L. A. Belady and M. M. Lehman. A model of large program development. IBM Systems Journal, 15(3):225-252, 1976.

[5] M. Bergquist and J. Ljungberg. The power of gifts: Organising social relationships in open source communities. Information Systems Journal, 11(4):305-320, 2001.

[6] A. Capiluppi. Models for the evolution of OS projects. In Proceedings of International Conference on Software Maintenance, pages65-74, Amsterdam, Netherlands, 2003.

[7] A. Capiluppi, P. Lago, and M. Morisio. Evidences in the evolution of OS projects through changelog analyses. In Proceedings of the 3rd Workshop on Open Source Software Engineering, Portland, OR, USA, 2003.

[8] A. Capiluppi, M. Morisio, and J. F. Ramil. Structural evolution of an open source system: A case study. In Proceedings of the 12th International Workshop on Program Comprehension (IWPC), pages 172-182, Bari, Italy, 2004.

[9] K. Crowston and J. Howison. The social structure of free and open source software development. First Monday, 10(2), 2005.

[10] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In Proceedings of International Conference on Software Maintenance, pages 23-32, Amsterdam, Netherlands, 2003.

[11] K. F. Fogel. Open Source Development with CVS. The Coriolis Group, Scottsdale, Arizona, 1st edition, 1999.

[12] D. M. German. An empirical study of fine-grained software modifications. pages 316-325, Chicago, IL, USA, 2004.

[13] D. M. German. Using software trails to reconstruct the evolution of software. Journal of Software Maintenance and Evolution:Research and Practice, 16(6):367-384, 2004.

[14] D. M. German and A. Mockus. Automating the measurement of open source projects. In Proceedings of the 3rd Workshop on Open Source Software Engineering, Portland, OR, USA, 2003.

[15] M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In Proceedings of the International Conference on Software Maintenance, pages 131-142, San Jose, CA, USA, 2000.

[16] J. Howison and K. Crowston. The perils and pitfalls of mining SourceForge. In Proceedings of the International Workshop on Mining Software Repositories (MSR 2004), pages 7-11, Edinburgh, UK, 2004.

[17] K. Johnson. A descriptive process model for open-source software development. Master's thesis, Department of Computer Science,University of Calgary, 2001. http://sern.ucalgary.ca/students/theses/KimJohnson/thesis.htm

[18]N. Jørgensen. Putting it all in the trunk: Incremental software engineering in the FreeBSD open source project. Information Systems Journal, 11(4):321-336, 2001.

[19] S. Koch and G. Schneider. Effort, cooperation and coordination in an open source software project: GNOME. Information Systems Journal, 12(1):27-42, 2002.

[20] M. M. Lehman and L. A. Belady, editors. Program evolution: Processes of software change. Academic Press Professional, Inc., San Diego, CA, USA, 1985.

[21] L. Lopez, J. G. Barahona, I. Herraiz, and G. Robles. Applying social network analysis techniques to community-driven libre software projects. International Journal of Information Technology and Web Engineering, 11(4):321-336, 2006.

[22] T. Mens, J. F. Ramil, and M. W. Godfrey. Analyzing the evolution of large-scale software: Guest editorial. Journal of Software Maintenance and Evolution, 16(6):363-365, 2004.

[23] M. Michlmayr. Managing volunteer activity in free software projects. In Proceedings of the 2004 USENIX Annual Technical Conference, FREENIX Track, pages 93-102, Boston, USA, 2004.

[24] M. Michlmayr. Software process maturity and the success of free software projects. In K. Zielinski and T. Szmuc, editors, Software Engineering: Evolution and Emerging Technologies, pages 3-14, Krakow, Poland, 2005. IOS Press.

[25] M. Michlmayr and B. M. Hill. Quality and the reliance on individuals in free software projects. In Proceedings of the 3rd Workshop on Open Source Software Engineering, pages 105-109, Portland, OR, USA, 2003.

[26] M. Michlmayr, F. Hunt, and D. Probert. Quality practices and problems in free software projects. In M. Scotto and G. Succi, editors,Proceedings of the First International Conference on Open Source Systems, pages 24-28, Genova, Italy, 2005.

[27] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology, 11(3):309-346, 2002.

[28] E. S. Raymond. The Cathedral and the Bazaar. O'Reilly & Associates, Sebastopol, CA, USA, 1999.

[29] A. Senyard and M. Michlmayr. How to have a successful free software project. In Proceedings of the 11th Asia-Pacific Software Engineering Conference, pages 84-91, Busan, Korea, 2004. IEEE Computer Society.

[30] N. Smith, A. Capiluppi, and J. F. Ramil. Agent-based simulation of open source evolution. Software Process: Improvement and Practice, 11(4):423-434, 2006.

[31] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris. Code quality analysis in open-source software development. Information Systems Journal, 12(1):43-60, 2002.

[32] L. Torvalds. The Linux edge. In C. DiBona, S. Ockman, and M. Stone, editors, Open Sources: Voices from the Open Source Revolution, pages 101-111. O'Reilly & Associates, Sebastapol, CA, USA, 1999.

Project Entity Matching across FLOSS Repositories

Megan Conklin
Elon University Campus Box 2126, Elon, NC, 27244 USA
mconklin@elon.edu
WWW home page: http://facstaff.elon.edu/mconklin

Abstract. Much of the data about free, libre, and open source (FLOSS) software development comes from studies of code repositories used for managing projects. This paper presents a method for integrating data about open source projects by way of matching projects (entities) and deleting duplicates across multiple code repositories. After a review of the relevant literature, a few of the methods are chosen and applied to the FLOSS domain, including a simple scoring system for confidence in pairwise project matches. Finally, the paper describes limitations of this approach and recommendations for future work.

1 Introduction

Free, libre or open source software (FLOSS) development teams often use centralized code repositories to help manage their project code, to provide a place for users to find the product, and to organize the development team. Although many FLOSS projects host their own code repository and tools, many projects use the tools hosted at a third-party web site (such as Sourceforge[1], ObjectWeb[2], or Rubyforge[3]). These code forges provide basic project/team management tools, as well as hosted space for the source code downloads, a version control system, bug tracking software, and email mailing lists. There are also directories of FLOSS software (such as Freshmeat[4] and the Free Software Foundation[5] directory) that try to gather into one convenient place material about projects interesting to a particular community.

Much software development research has been focused on gathering metrics from code repositories. Many aspects of the repository-based software development

[1] http://www.sf.net
[2] http://forge.objectweb.org
[3] http://www.rubyforge.org
[4] http://www.freshmeat.net
[5] http://directory.fsf.org

process have been studied in depth, and repository data collection is important for these studies (see [2] for background). The FLOSSmole project [5] was created to consolidate metadata and analyses from some of these repositories and directories into a centralized collaboratory for use by researchers in industry and academia. As of this writing, FLOSSmole includes data and analyses from Sourceforge, Freshmeat, Rubyforge, ObjectWeb, and the Free Software Foundation (FSF) directory of free software. One of the challenges mentioned in [2] in creating this kind of collaboratory is in integrating the data from these various sources. When integrating project data from multiple sources, we must first identify which project pairs are matches; in other words, we want to find out which projects are listed on multiple forges. For example, is the *octopus* project on ObjectWeb the same as the *octopus* project on Sourceforge or the project also called *octopus* on Freshmeat? If we can determine a heuristic for determining whether a project pair is a match, then can we automate the matching process?

The focus of this paper is entity matching (and duplicate identification) as applied to the domain of FLOSS projects. Section 2 outlines some terminology from the study of data integration problems and gives a background of entity matching algorithms. Section 3 describes the FLOSS domain in terms of entities and duplicates. Section 4 gives an example of applying some of the algorithms for entity matching to this domain. Section 5 outlines limitations of this work and gives recommendations for future study.

2 Entity Matching Background

The act of integrating multiple data sets and finding the resulting duplicate records ("matches") is nearly as old as database processing itself. In practice and in the literature, this set of processes is known by many names: merge/purge, object identification, object matching, object consolidation, record linkage, entity matching, entity resolution, reference reconciliation, deduplication, duplicate identification, and name disambiguation. The terms *entity matching* and *duplicate identification* will be used throughout this paper.

Within the larger activity of data integration, the act of matching entities or identifying duplicates is not to be confused with the act of *schema reconciliation*. Schema reconciliation refers to the act of matching up columns or views in different data sources, and using data or metadata to make the match. For a trivial example, suppose a field in Table A is called *url* but it is called *home_page* in Table B. To resolve these schemas, the analyst could create a global schema or view that encapsulates both underlying schemas. This task can be done manually, or can be automated through various machine learning techniques [1,4,8]. Schema reconciliation and entity matching are related, but not identical, tasks of data integration. Most often the schema reconciliation will happen first, followed by the "merge" task, and finally by the eventual "purge" of duplicate data.

2.1 Agree/Disagree and Frequency-Based Matching

The simplest and oldest form of entity matching is the simple agree/disagree method: take two data sets A and B and compare them pairwise for matches based on one or more attributes. The pairs will either agree or disagree on zero or more of the attributes, and thus a weight for the match can be determined.

To improve agree/disagree entity matching, early research relied on frequencies of values to determine the probability of a match (see [10] and [7] for brief explanations of this work). Frequency matching asserts an important premise: that two rare values are more easily and accurately matched than two common values. The example given in [10] is for two records listing the name Zbigniew Zabrinsky, two records listing the name James Smith, and any two records with first and last names. The two records for Zbigniew Zabrinsky are likely to be more easily and accurately matched than James Smith due to the rarity of the field values.

2.2 Disjoint Sets

In [4], the authors consider the problem of how to match 'person' records using disjoint attributes and a 'typical person' profile. For instance, the example given in the paper is that the two records {Mike Smith, age 9} and {Mike Smith, salary $200,000} are not likely to be the same person based on a profile indicating that a typical person with an annual salary of $200k is older than 9 years. The authors compare their system to a traditional agree/disagree system of matching, and show that disjoint attributes can be effective if paired with shared attributes.

2.3 Confidences

Numerous authors, including recently [6], consider how to merge records when a confidence measure has been added to the results of a prior merge process. In their description, confidences (also called weights or scores) usually measure either (a) the level of user-defined "belief" in the data, or (b) the amount of "accuracy" the user thinks is present in that particular merged record. In their paper, the authors ask: what is the best (most efficient, least work) way to match and merge records, given a confidence measure on each record? These authors do not discuss in [6] how to actually calculate a confidence value, but this is one of our concerns in Section 4.

3 Entity Matching Methods for FLOSS Data

This portion of the paper describes the way each of these entity matching methods can be applied to integrate disparate sets of projects from the FLOSSmole project.

First, by way of introduction to the FLOSSmole data, Table 1 shows a partial list of the project attributes available for each of the repositories/forges in FLOSSmole at the time of this writing. These project attributes are the most likely candidates for the job of matching projects. (There are dozens of other attributes about each project in FLOSSmole, such as *registration date* or *project status* or *number of downloads*, but these are not likely to be helpful in matching projects across repositories.)

Attribute	**Forge** *Sourceforge, Freshmeat, Rubyforge, Objectweb, Free Software Foundation*				
	SF	FM	RF	OW	FSF
Short Name (unixname)	X	X	X	X	X
Long Name	X	X	X	X	X
Description	X	X	X	X	X
URL	X	X	X	X	X
License Type(s)	X	X	X	X	X
Programming Language(s)	X	X	X	X	X
Operating System(s)	X	X	X	X	
Topic(s)	X	X	X	X	
Intended Audience(s)	X	X	X	X	
User Interface(s)	X	X			X
Environment(s)			X	X	
Developer(s)	X	X	X	X	X

Table 1. Project metadata: relevant attributes for matching projects, (FLOSSmole, Dec, 2006)

Most of these attributes shown in Table 1 are self-explanatory. However, some confusion can arise when differentiating between the *short name* and the *long name* for a project. The short name is usually an internal-to-the-repository name that is given to the project at the time of its creation. Some repositories use this as a sort of primary key for the project in its database. The long name of a project is the more descriptive name for a project. It can change over time, it can include spaces and special formatting characters, and it typically more descriptive than the short name. Values for all of the attributes shown in the list in Table 1 are chosen by the project administrators, and except for short name and long name, they can all be NULL. License type, operating system, topic, audience, interface, and environment can have multiple values.

The next three sections describe a few of the obvious choices for attributes from this list that can be used to establish matches between projects. One choice from Table 1 that may initially look promising is "List of Developers". Since this attribute is actually a list of developers who work on each project, what better way to differentiate or match two projects? (If the list of developers is the same for the two projects, then the two are a match.) The problem with this is that developers are entities themselves, and matching developers between repositories requires an entirely separate list of attributes (developer name, developer email, developer skills, role on project, etc). Section 5 discusses broadening project entity matching to include developers, but the remainder of this paper will exclude developers as entities and will retain the focus on project matching only.

3.1 Matching by URLs

The diagram shown in Figure 1 depicts each forge/directory in FLOSSmole and how many of its projects list *another forge* as the actual hosting home page. For example, in the diagram, the topmost arrow shows 11 projects on the FSF that actually have Rubyforge listed as the home page. The arrow notation is used to show a direction of the relationship (e.g. 10,044 Freshmeat projects show a home page on Sourceforge, but only 4 Sourceforge projects list a Freshmeat home page). Pairs of forges with no URLs in common are not shown. (No Rubyforge projects list ObjectWeb URLs, and vice versa. Also, as is befitting its status as a directory and not a repository, the FSF directory is not listed as the home page of *any* projects from the other repositories, so these empty relationships are not shown in the diagram.)

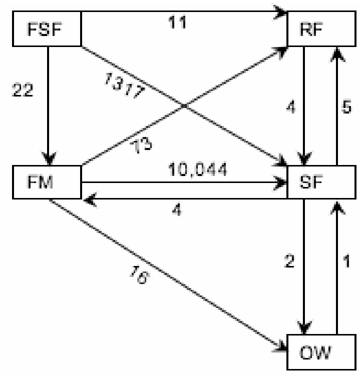


Fig. 1. Number of projects at each repository that list a home page at another repository

3.2 Matching by Project Names

Figure 2 shows the number of short project names shared in common between each pair of projects. For instance, *starfish* is a project listed on both Sourceforge and Rubyforge. On Rubyforge, it is described as a "tool to make programming ridiculously easy", but on Sourceforge the *starfish* project is described as a password management application. There are 470 projects with shared names on Rubyforge and Sourceforge. A similar problem exists between the project names on Sourceforge and ObjectWeb. For example, the project called *octopus* exists on both these forges and appears to be a completely different application: on Sourceforge this is an Eclipse plug-in, but on ObjectWeb *octopus* is an ETL data warehousing tool. Of the 125 applications (total) listed on ObjectWeb, 41 have names that are shared with a Sourceforge project. The Sourceforge project may (as in the case of *lemonldap*) or may not (as in the case of *octopus*) be the same project. On Freshmeat, there also is a project called *octopus*, but this one is a financial trading application.

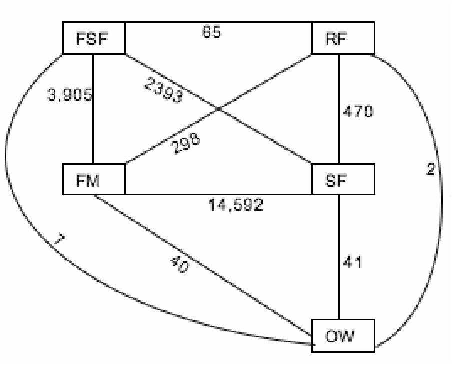


Fig. 2. Number of projects at each repository that share an identical short project name

Most forges require projects to have a unique name (sometimes called the "unixname") within that forge. For example, once a project called *starfish* has been added to Sourceforge, another one cannot be added with the same short unixname. However, multiple projects can have the same "display name"; Sourceforge projects *starfish* and *xstarfish* both have the display name of "starfish". On Sourceforge, 44,112 (39%) of projects have unixnames that are different from their display names (December 2006 FLOSSmole data). Note that the FSF directory has only a requirement for case-sensitive uniqueness in project names. The FSF lists project pages for both ANT (telephony application) and ant (build tool). There are 54 such (ambiguously) named projects listed on FSF.

3.3 Matching by Other Attributes

It may be possible to determine the accuracy of each matched pair further by attempting to match the project owner or developer names, emails, or usernames as in [11]. Or, it may be possible to find a matched pair through the textual description of the project, or through the project license type, the programming language(s), operating system(s), or other metadata about the project. Each of these possible match fields requires that the project administrator has accurately filled in the metadata for his/her project. If the administrator never bothered to fill in the programming language for the project on one or both of the sites where the project is listed, then it will not be possible to disambiguate by finding a match on this item.

Project Attribute	**Projects listing at least one**	**Projects listing none**
Programming Language	82,969 (73%)	29,946 (27%)
License Type	84,102 (74%)	28,813 (26%)
Operating System	78,334 (69%)	34,581 (31%)

Table 2. Numbers of Sourceforge projects with and withoug certain attribute data, (FLOSSmole, Dec, 2006)

Table 2 summarizes a few of the most common attribute statistics for Sourceforge projects. It is also interesting to discover that of those 74% of projects that list a license type, over half use the GPL.

3.4 Advanced Methods

In our attempt to match FLOSSmole projects by URL, name, or any other combination of attributes, we are still performing basic agree/disagree entity matching. Our brief review of the database literature on entity matching indicates that these methods do work for some cases, but can be optimized and improved.

3.4.1 Frequency-Based Matching

The first improvement made to the agree/disagree entity matching was to consider how to apply a form of frequency matching on the name field. Recall that [10] explains that rare names (Zbigniew Zabrinsky) are more easily matched than common names (James Smith). "Rare" and "common" are determined by an already-existing set of names and their general frequency rankings in the population. In the case of FLOSS projects, there is no such ranking for software project names, but a corollary might be that projects with dictionary words for names (e.g. the *octopus* and *starfish* examples) are more likely to be non-matches than projects with unusual, non-dictionary names (e.g. *sqlite-ruby* or *lemonldap*). Because there is also a difference between the unique *unixname* and the non-unique *display name* for each project, we ask: which of these fields should be used to consider the frequency match? In Section 4, we answer with "both", but we score the matches differently.

3.4.2 Disjoint Sets

The next improvement was to use the notion of a disjoint set, as in 2.2. by listing which attribute values would likely *never* coexist. Initial ideas included the following possible disjoint sets: {*op_sys*=linux, *prog_lang*=asp}, {*date_registered*<2001, *prog_lang*=C#}. Not only are these rules fairly weak insofar as there are plenty of examples of projects that would violate them for various reasons, but unlike the age/salary information in the example case in 2.2, the number of records in FLOSSmole which match these disjoint sets is likely to be quite small. We conclude that in the FLOSS domain, it is more likely to be the case that duplicates can be found through simpler methods than disjoint sets. This is due to three factors: the low number of valid disjoint set rules we would be able to construct, the difficulty of applying disjoint set rules to our data when so many of the pairs are missing metadata on which these disjoint sets would be based (recall section 3.3), and the low number of duplicates that *would not* be identified by other, simpler methods.

3.4.3 Confidences or Scoring

One final serious consideration in advanced methods of entity matching and duplicate identification was the use of confidences to describe numerically the analyst's degree of belief in the accuracy of the merge/match. How should scoring be done? We worked backwards from our initial assumption that the end goal of this exercise is be able to point from one project to another based on likelihood that they are a match. Thus, we planned to consider each pair in turn, then apply a confidence/match score (based on the heuristics used) to the record to indicate how good the match was. The scoring and results are given in the next section.

4 Application

To apply entity matching methods to project data in FLOSSmole, we assume a set of heuristics and associated weights for calculating whether the items in a pair are a match (Table 3). Match modifiers were determined through trial and error, and based on an intuitive sense of which matching criteria were important.

	Match Modifier
Home Page URLs match	+3.00
Short names match	+2.00
--if yes, is short name in the dictionary (i.e. is it common?)	-1.00
--if not, does Partial Name match?	+0.50
-- if partial name matches, is partial name in dictionary?	-0.25
Textual descriptions tokens match, per token match	+0.10
Long (display) names match	+0.50
Programming language matches, per token match	+0.50
License matches, per token match	+0.50
Other project metadata matches, per token match	+0.50

Table 3. Scoring table for matching pairs of projects

A short example of a table designed to hold the FLOSSmole pairs with their matching scores across multiple repositories might look like Table 4. Higher scores mean the pair is more likely to be a match, but it will be up to an individual analyst to decide where to "draw the line" for what score indicates a match. The highest score is around 8; the lowest score is 0. As shown in the table, the highest score could be higher if more attributes were added. Attributes included are programming language, operating system, and license because these are the fields whose values were most available and easiest to standardize over a variety of repositories. (Compare with attributes like "environment", "interface", or "topic" that are hard to standardize.)

Pair ID	Project Name	Source A	Project Name	Source B	Score
1	phpmyadmin	SF	8001 (phpmyadmin)	FM	6.9
2	octopus	SF	octopus	OW	1.0
3	octopus-ge	SF	octopus	OW	2.6
4	16120 (octopus)	FM	octopus	OW	1.5
5	13902 (ant)	FM	152 (ant)	FSF	4.1
6	sqlite-ruby	SF	sqlite-ruby	RF	6.9

Table 4. Scoring table for matching pairs of projects

Pair 1 shows Sourceforge project *phpmyadmin* matching an identically-named Freshmeat project. These projects share a short name (with low frequency count when compared to a dictionary word: +2), long name (+.5), URL (+3), and license type (+.5). Several key tokens are the same in each description (+.9: MySQL, PHP, Web, administration, alter, drop, database, delete, SQL).

Pair 2 shows Sourceforge project *octopus* with ObjectWeb project *octopus*. The short project names match (+2), but urls are different. The long project names are also different ('Octopus' and 'Enhydra Octopus'). Additionally, because the Sourceforge project *octopus* does not list any project metadata, it can't be matched very well with the ObjectWeb project of the same name using these additional attributes. Finally, these two entities share the dictionary name 'octopus' (-1).

Pair 3 shows the project *octopus-ge* on Sourceforge and project *octopus* on ObjectWeb. These projects share a beginning partial string match, octopus* (+1) but it is a dictionary word (-.5). They share one programming languages (+.5), a license type (+.5), and one operating system (+.5). The textual description of the projects increases the score, since both use the strings 'Enhydra Octopus', 'extraction', 'transformation', 'load*', 'ETL', and 'XML' (+.6). However, a closer read of the textual description field by a human being reveals that the Sourceforge project is actually a graphical editor for the ObjectWeb project. They are related projects, but not the same project. The combination of no ULR score and low scores for the textual matches has (accurately) kept this project from a high score.

Pair 4 shows the attempted match between that same *octopus* project at ObjectWeb but now paired with the *octopus* project at Freshmeat. The projects have the same short name (+2 for similarity, -1 for dictionary), but different URLs, totally different textual descriptions, and share only the license type in common (GPL, +.5). Indeed, manual checking of this result shows that these two projects are not related.

Pair 5 shows the Freshmeat project *ant* matching with the Free Software Foundation project *ant* as follows: short name (+2), url (+3). However, the display names for this project are different ('ant' on FSF and 'apache ant' on FM). In addition, the common dictionary name 'ant' lowers the score somewhat (-1). Note that while there is only one significant matching token in the textual description (the word

"Java", +.1), the entire first sentence of the two projects is identical. This indicates a strong need to refactor the scoring algorithm for textual descriptions.

Pair 6 shows that SQLite-ruby project listed nearly identical information on both Sourceforge and Rubyforge. They share: the project home page (+3), short name (+2), the display name (+.5), one programming language (+.5), the operating system (+.5), and 4 significant text tokens (+.4), yielding a total score of 6.9.

5 Limitations, Recommendations, and Future Work

Based on the application shown in Section 4, entity matching is an interesting exercise, but is certainly problematic. One of the most obvious problems is the scoring modifiers given in Table 2; there is a distinct possibility that a pair of projects could achieve a score of 4.0 by having a partial non-dictionary name match (+.5), five attributes in common (+2.5), and a handful of well-chosen tokens in the textual description (+.5), and yet these projects could be completely unrelated. Yet, it is not enough to simply require a score higher than 5.0 for a match; according to the table, the *ant* project pair on Freshmeat and FSF also received a score of 4.1, and it *is* a legitimate match.

This leads to a discussion of how to set scoring thresholds. Perhaps there could be a "yes" category for projects scoring above a certain value, a "no" category for projects scoring below a certain value, and a middle category for questionable scores. These questionable scores may take human intervention to resolve. It will be necessary to constantly tweak the scoring system and thresholds so that there are not too many false positives, false negatives, or values needing human intervention.

There are also numerous ways to improve the definitions of token matches within textual descriptions. For instance, in the case of *ant*, there were very few singularly meaningful tokens in the textual descriptions, but the description as a whole matched perfectly. The use of dictionary word definitions for frequency matching may need to be refactored also. The *ant* match lost points because of this. Also, should non-dictionary strings that are also common in software development ("lib", "db", "php") be added to the dictionary? Partial matches were also problematic. How should the word be broken: by leading strings, ending strings, or middle-of-word strings? Also, if a project name matches by 14 letters, should that get a higher score than a pair that only matches by three letters? Is it possible that those three letters could be highly significant?

Next, what about multi-way matches? We have given little attention to the problem (as presented in [6]) of how to merge multiple confidence scores after they've been created. Consider a project such as *sqlite-ruby* that appears on Sourceforge, Rubyforge, Freshmeat, and the FSF directory. What is the appropriate way to integrate its multiple scores? *Sqlite-ruby* is likely to have high scores on all 6 pair combinations, so a simple average might work, but what about a project like *ant* whose scores may vary more?

Section 3 mentioned the possibility of matching projects based on the lists of developers on each project. Before doing this, it would be necessary to use similar entity matching methods to actually match *developer* entities as well. As is so well-described in [9], matching developers also leads to a few additional complexities: "real" emails are most often not available for public lists of developers on code repositories, name matching with developers could be even more complex than matching on names for projects because of similarities in names and spellings, and of course, developer privacy is always a concern when integrating disparate personal data. It is instructive that the authors in [9] do also rely on heuristics to make their matches, and that they limit their matches to a single group of actors in the GNOME project, albeit over numerous data sources within that project (mailing lists, CVS repositories, etc.)

One final recommendation for future work is to remember some of the work being done on sites like Krugle[6], Swik[7], and the Galactic Project Registry[8] to standardize the notion of a project name. Krugle is a source code search engine that actually uses some FLOSSmole data to populate its list of projects. Swik is a wiki of information about individual open source projects; it gets some of its initial information from FLOSSmole as well. The Galactic Project Registry is attempting to put together a plan for being "the One True Known Up-To-Date Source" for project names and DOAP (description of a project) information on each project. Each of these projects probably would benefit from this work in entity matching and duplicate identification across repositories, and perhaps they can contribute to the conversation about the best way to achieve this goal.

6 Acknowledgements

The author would like to acknowledge Charles Irons and Wayne Conley for their important work on the *Other Souths* project at Elon University. Their work implementing *Other Souths* directly influenced the methods designed for this paper.

7 References

1. Batini, C., Lenzerini, M., Navathe, S. (1986). A comparative analysis of methodologies for database schema integration. *ACM Comp. Surveys*, 18:4. 323-364.

[6] http://www.krugle.com
[7] http://www.swik.net
[8] http://gpr.wikiwall.org

2. Conklin, M. (2005). Beyond low-hanging fruit: Seeking the next generation of FLOSS data mining. In *Proc. 2nd Intl. Conf. on Open Source Sys.* Como, Italy. 47-56.

3. Doan, A., Domingos, P., Halevy, A. (2001). Reconciling schemas of disparate data sources: A machine learning approach. In *Proc. of the ACM SIGMOD*. Santa Barbara, CA, USA. 509-520.

4. Doan, A., Lu, Y., Lee, Y., Han, J. (2003). Object matching for information integration: A profiler-based approach. In *Proc. of the IJCAI Workshop on Information Integration on the Web*. Acapulco, Mexico. 53-58.

5. Howison, J., Conklin, M., Crowston, K. (2005). OSSmole: A Collaborative Repository for FLOSS Research Data and Analyses. In *Proc. of the 1st Intl. Conf. on Open Source Sys*. Genova, Italy. 54-59.

6. Menestrina, D., Benejelloun, O., Garcia-Molina, H. (2006). Generic entity resolution with data confidences. In *Proc. of 1st Int. VLDB Workshop on Clean Databases*. Seoul, Korea.

7. On, B-W., Lee, D., Kang, J., Mitra, P. (2005). Comparative study of name disambiguation problem using a scalable blocking-based framework. In *Proc. of the 5th ACM/IEEE-CS Joint Conf. on Digital Libraries*. Denver, CO, USA. 344-353.

8. Rahm, E. and Bernstein, P. (2001). A survey of approaches to automatic schema matching. *VLDB Journal*, 10. 334-350.

9. Robles, G. and Gonzalez-Barahona, J. (2005). Developer identification methods for integrated data from various sources. In *Proc. of the Mining Software Repositories Workshop (MSR2005)*. 1-5.

10. Winkler, W. (1999). The State of Record Linkage and Current Research Problems. Technical Report, Statistical Research Division, US Bureau of the Census.

Can Knowledge Management be Open Source?

Charmaine C Pfaff[1] and Helen Hasan[2]
1 University of Wollongong, Faculty of Commerce, Information Systems, Wollongong 2522, Australia
hasan@uow.edu.au
WWW home page: http://www.uow.edu.au/commerce/infosys/hasanh.html
2 University of Wollongong, Faculty of Commerce, Information Systems, Wollongong 2522, Australia
ccp02@uow.edu.au

Abstract. As we move further into a knowledge economy where collaboration and innovation are increasingly central to organisational effectiveness, enterprises need to pay more attention to the informal networks that exist within the organisation. Wikis may provide a more appropriate knowledge management capability and environment to capture tacit knowledge. Where traditional organisational cultures see that knowledge management must be tightly protected, Wikis opt for an open source approach where knowledge is shared and distributed for innovation to continue. This paper aims to explicate more participatory organisational processes of creation, accumulation and maintenance of knowledge. It uses Activity Theory as a framework to describe the components of an activity system where a Wiki is a tool mediating employee-based knowledge management activities and thereby democratising organisational knowledge.

1 Introduction

As organisations aim at moving knowledge from the realm of the individual into the hands of the organisation, they often resort to expensive Knowledge Management Systems (KMS) with data mining and search engines to organize and access large volumes of documents. Although traditional business logic dictates that there must be organisational controls to ensure conformity so that tasks can be defined and measured, they stifle creativity and initiative, constraining the design of the next generation KMS. In practice, the new business environment requires a KMS that performs better on fewer rules, some specific information and greater freedom. The goals of this paper are to analyse the essential elements of organisational

Please use the following format when citing this chapter:

Pfaff, C.C. and Hasan, H., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 59–70.

knowledge, KM and knowledge workers in creating a more cooperative and democratic KM and the potential of using corporate Wikis as new generation KMS.

2 Managing Knowledge and Knowledge Workers

2.1 The nature of knowledge

The current corporate interest in knowledge is based on a realisation that emerging economic imperatives, coupled with social and industrial restructuring, demand a more rigorous approach to the exploitation of knowledge as an organisational resource. Organisational knowledge can be about what employees understand about historical knowledge inherent in the organisation such as the knowledge about customers, products, processes, errors, and successes. Various streams of KM research have emerged. However, the differences in interpretation and definition have become a matter of contention.

It is challenging to scan the human mind for tacit knowledge (knowledge that is embedded in a person's mind and cannot be expressed easily and explicitly) because most individuals may know more than they think they know. The sense meaning making capacity of the human mind may evoke tacit knowledge as a response to new and unfamiliar stimuli or situations that may not fit previously recognised scenarios. In addition, it ignores the possibility that additional knowledge resides in the relationships between employees and in the legacy of previous employees embedded in organisational memory and culture.

From an IS perspective, knowledge is the top of the data-information-knowledge hierarchy where information is meaningful, processed data and knowledge is actionable information, separating knowledge from information or data (Handzic & Hasan 2003). This view of knowledge reinforces the value of using CHAT for research on KM because it can extract *actionable meaning* from unstructured or ill-structured information, social interaction patterns, and deep rooted motives of knowledge workers.

2.2 Problems with managing knowledge

Many KMS have not met their original business objectives because there is an assumption that all relevant knowledge, including tacit knowledge should be extracted from knowledge workers and stored through well-established institutional processes in well-designed knowledge repositories (AS5037[Int] 2003). The process of building these repositories has been criticised as being time-consuming, laborious, and costly. The process of building these repositories has been criticised as being time-consuming, laborious, and costly. Viewed by many as a superficial implement of management, they are often not kept up-to-date and are rarely accessed when real knowledge is sought (Klint & Verhoef 2002).

Hart & Warne (2005) have stressed that it is detrimental to manage organisational knowledge because knowledge by its very nature cannot be managed in the traditional sense. KM cannot be fostered in settings where people feel pressured as it makes them less motivated to engage in dialogue. Often, employees hoard their knowledge because their contributions do not benefit their careers and becomes an additional burden to their already heavy workloads (Lam & Chua 2005). Hence, the authors support the Australian Standard (AS 5037—2005) definition of KM: "KM is concerned with innovation and sharing behaviours, managing complexity and ambiguity through knowledge networks and connections, exploring smart processes, and deploying people-centric technologies."

The focus on work practices reveals how community members conceptualise the work they perform and the synergistic roles of the community and its members in the processes of knowledge production. 'Knowledge work' is not restricted to the work practices of individuals and teams that create and exploit knowledge (Burstein & Linger 2003), knowing 'how' and 'who' you know are as important as what you know. Understanding how knowledge workers work and their needs will help them become more productive.

3 The Wiki Way

3.1 Open source revolution

The notion of creative collaborative work is not new. Its best known propagator is the open source software development. The open source movement (OSM) began as experiments in software democracy that crossed institutional and geographical boundaries. It has achieved a momentum in motivating people to work together in self organised groups on common projects and making them available on the Internet for use or modification. The OSM is fulfilling the original promise of the Internet and promoting the Internet culture where people can work together in an environment that supports access to information. Already, it has inspired the emergence of an ecosystem of other projects such as Creative Commons sharing media resources and *Wikipedia*. KMS can learn some lessons from its success such as simplicity in design, frequent reviewing and testing, a skilled and devoted group of volunteers and developers, and simple but effective rules to govern the community (Wagner 2006). It is our contention that new ICT tools such as the corporate Wiki can be the enabler to effect changes for the better in organisations. For example, organisations that adopt a rigorous 'best practices' approach find it extremely challenging not to be caught in the death spiral (Nadler and Shaw 1995) of doing *more of the same better and better* with diminishing marginal returns (Drucker 1994). The corporate Wiki provides an environment to ensure that such practices remain open to critique, adaptation, and replacement.

3.2 Conversational Technology

New conversational technologies such as email, discussion forums, chatrooms, Weblogs and Wikis are now connecting and supporting liberated knowledge exchanges much as transportation systems and cities on the ground have always done. Interconnected networked structures of social interaction and creative activity are emerging as a part of the civil digital culture and, less rapidly, in the knowledge work of organisations. Conversational technologies are seen as tools to support work units and the individual knowledge worker. It is the corporate Wiki that is of most interest to the field of KM because it can be developed by end users through collaboration (Hasan & Pfaff 2006a, Wagner 2006).

A Wiki is a web-based application that allows many participants to write collaboratively, where anyone can start a new page or edit an existing one. Such documents can be supported by the web with hyperlinks to anywhere on the World Wide Web including text, image and video. 'Wiki Wiki' in Hawaiian means 'quick' or 'fast' which refers to the quick editing processes (Leuf & Cunningham 2005). A Wiki is a collection of interlinked HTML web pages. Changes are logged and viewed online instantly and can be reverted to its original state. A Wiki can be accessed from any web browser and no other special tools are needed to create and edit existing pages. A Wiki is an evolving knowledge repository where users are encouraged to make additions to this repository by adding new documents or working on existing ones (Pfaff & Hasan 2006). The most well known example of a Wiki is Wikipedia[1], an online encyclopaedia exemplifying the open source ideal.

3.3 Factors contributing to the rise of *Wikipedia*

The openness of *Wikipedia* as a publicly editable website is a phenomenal motivating factor for people to work together and share their knowledge to teach the world. Emigh and Herring (2005) compared traditional printed sources with *Wikipedia* articles and found them stylistically indistinguishable and citing *Wikipedia* articles in news and other media have become common (Lih 2004). Wiki users feel a sense of ownership when they see their work online and want to "collaborate radically", a feature of the OSM where anyone can edit another person's work. Collaboration avoids bottleneck complications if there is an individual author and the constant editing refines the article (Sanger 2005). The neutral policy of a Wiki allows everyone to air their views while at the same time respecting divergent views.

Nupedia is the predecessor to *Wikipedia* which began in 2000. Its founders wanted volunteers to write, edit and review articles as they would for a printed, for profit published encyclopaedia. Nupedia was terminated in 2003 because of its server problems, intolerance to writers who are not experts and mainly, its complicated review process. (Rosenzweig 2006, Sanger 2005). The founders of *Wikipedia* reduced the turnaround time to edit and review *Wikipedia* articles, to overcome the participatory problem.

3.4 *Wikipedia* criticisms

The main allegation about *Wikipedia* is that the information varies in quality. To investigate this claim, Rosenzweig (2006) compared 25 *Wikipedia* biographies against comparable entries in *Encarta* and *American National Biography Online*. Although both publications have multimillion-dollar budgets, it was found that

[1] http://en.Wikipedia.org

Wikipedia articles were favourably written. *Wikipedia* is also accurate in reporting names, dates, and events in U.S. history. *Wikipedia* surpasses *Encarta* but not *American National Biography Online* in coverage and matches *Encarta* in accuracy. In another study, a German computing magazine engaged experts to compare *Wikipedia* articles in 22 different fields in the three leading German-language digital encyclopaedias. It rated *Wikipedia* first with a 3.6 on a 5-point scale, *Brockhaus Premium* scored 3.3 and *Encarta* 3.1. (Kurzidim 2004). A British scientific magazine, *Nature,* asked academic scientists to do a blind review of 42 science entries in *Wikipedia* and *Encyclopaedia Britannica*. *Wikipedia* contained around four inaccuracies and *Britannica*, had three (Giles 2005).

3.5 Adapting the Wiki in a corporate setting

Looking back at other technologies e.g. personal computers, email and instant messaging that enjoyed widespread popularity; management needs to think of how to adapt already popular social tools for corporate use because the impact of grassroots marketing should not be overlooked. Nevertheless, some of the problems facing *Wikipedia* are reflected in corporate Wikis. (Hasan & Pfaff 2006b, Wei et. al 2005). The principal dilemma of a Wiki is that, while its anarchic nature is desirable for fostering open debate without censorship, it raises questions whether the information is authoritative and credible, thus inhibiting its usefulness. Yet a critical factor to bear in mind is that *Wikipedia* is a public online Wiki. Employees who make contributions to the corporate Wiki are employed by the organisation as specialists whose opinions will be highly regarded by their organisations as trusted and authoritative.

The informal network approach that is currently favoured in a Wiki, may make some companies believe that their data quality will be affected and that system errors will occur. Their centralised and highly structured environment will make it difficult to adopt a 'community approach' towards knowledge acquisition. The problems of irresponsible behaviour and accountability issues for fraudulent data can be avoided because employees using a corporate Wiki will not be using "handles" but their real names to login to edit the Wiki. This means that every post or edit could be attributed to an individual employee. A footnote can be included to remind employees that usage could be traced back to them to deter intentional misuse. Wikis have a rollback feature which could be used by administrators to repair deletions or misuse. Daily backups can preserve the Wiki database against loss of data in case of system failures (Auger et. al. 2004).

Employees should not see the corporate Wiki as merely an online shared space and owned by a gatekeeper. The corporate Wiki should instead be seen as an open community process that encourages multiple iterations in the creation of a knowledge repository (Wei et al 2005).

4 Cultural Historical Activity Theory (CHAT)

4.1 Theoretical basis

Although CHAT was proposed long before the advent of computers and the Internet, a growing band of researchers recognise that CHAT provides a rich holistic understanding of how people collaborate with the assistance of sophisticated tools in the complex dynamic environments of modern organisations (Thomas & Torstein 2005, Waycott et. al 2005, Hasan 1999).

The notion of activity is interpreted from the theory of Leontiev (1981) which is based on Vygotsky's psychology. Vygotsky (1978) proposed that all human activity is purposeful, carried out through the use of 'tools' and socially mediated. Tools can manipulate and transform objects but also restrict what can be done within the limitation of the tool, which, in turn, often stimulates improvements to the tool (Verenikina & Gould 1998), especially in the context of analysing the dialectic interactions between people and technologies, and how they are shaped by human activity.

4.2 Assumptions of the Activity System

The analysis begins with the identification and explication of the central activity and then looks at those activities that are linked to it (Hasan 2003a). As described in Hasan (2003b), Engeström (1987), Kuutti and Virkunnen (1995), an activity system normally has one central activity, which is the focal point of holistic investigation, surrounded by other activities with some link to the central activity.

Figure 1 shows the *activity* as the engagement of a subject toward a certain goal or objective where the project team is a *collective subject* composed of individuals who bring different skills and understandings to bear on a *common object*, the corporate Wiki. The purpose of the Wiki activity is to create, share and manage knowledge in the form of an encyclopaedia, which will persist over time while the participants may change. The core activity (object), for which a corporate Wiki is used, is not KM per se but knowledge work. There is a dialectic relationship between knowledge and work, expressed by the continuous cycle of co-creating work-related knowledge in a form that is meaningful for knowledge workers to access as needed, through which learning occurs, resulting in more knowledgeable doing and so on.

The tools are the Wiki technology together with social and learning processes within the organisation. Each participant subject brings different personal characteristics that may change over time, including individual motivations, goals, and self perceptions affecting the transformation of goals. Contributions can come from users' personal knowledge, which is related to fields where they feel comfortable and competent such as work projects or knowledge specialisations.

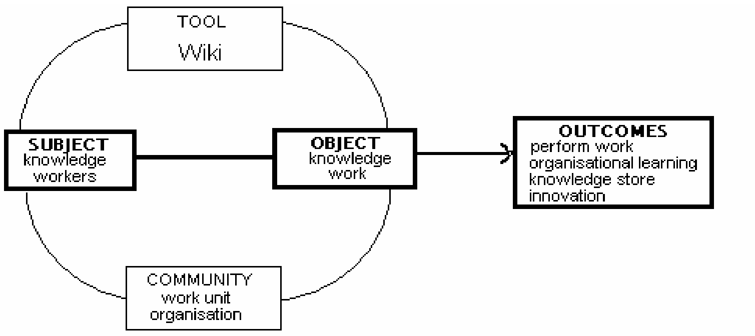


Fig. 1. The core activity of knowledge work mediated by a corporate Wiki

In trying to address the limitations of prior KMS, a corporate Wiki overcomes the barrier of KMS created from the static accumulation of dynamic knowledge. The activities of the knowledge worker are mediated not only by the functions of the corporate Wiki itself, but also by the attitudes and customs of the organisations in giving workers the resources and authority to do so. As knowledge workers operate at the grass roots level, they are in the best position to act as sense makers in a rapidly changing dynamic environment. Knowledge workers can participate as writers and peer reviewers, giving them opportunities to define problems and generate their own solutions, evaluate and revise their solution-generating processes.

Managers can embrace change by building up individuals, and reducing the dependence on strengthening institutions, which in turn has led to the creation of a highly protective culture of the status quo.

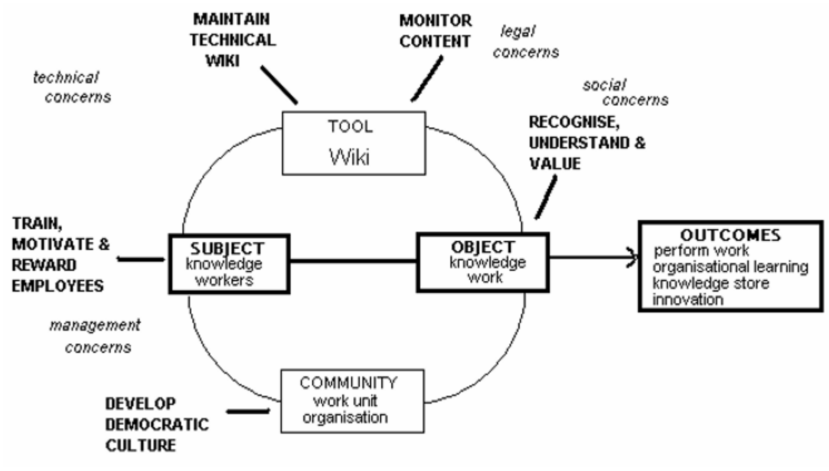

MAINTAIN TECHNICAL WIKI
MONITOR CONTENT
legal concerns
technical concerns
social concerns
RECOGNISE, UNDERSTAND & VALUE
TOOL Wiki
TRAIN, MOTIVATE & REWARD EMPLOYEES
SUBJECT knowledge workers
OBJECT knowledge work
OUTCOMES perform work organisational learning knowledge store innovation
management concerns
COMMUNITY work unit organisation
DEVELOP DEMOCRATIC CULTURE

Fig. 2. Elements of secondary activities related to knowledge work using a corporate Wiki

The distinctive attributes of corporate Wikis give us an indication of the technical, legal, social and management concerns that links to the core knowledge work activity shown in Figure 2. Technical concerns include ensuring that there is technical support to install the corporate Wiki and maintain overall quality. The legal aspects such as copyright issues and legal liability will increase the social responsibility of Wiki users as time progresses. The very nature of a Wiki is democratic as it values participation over power. Knowledge workers can only thrive in a culture that respects the minds of knowledge workers and promotes experimentation and rethinking. Democracy raises public awareness of issues such as openness, freedom of information and public accountability (Benkler 2006). A corporate Wiki will redefine the social constructs of the organisation because it places knowledge workers on the same level as management. The main hurdles for management are issues of trust, credibility and authority. Management has been concerned that previous KMS failed to motivate employees to contribute and use its contents (Hasan 2003a). As demonstrated by the popularity of *Wikipedia*, the corporate Wiki can overcome this participatory problem. However, this may depend on the organisation having a knowledge sharing culture and appropriate job descriptions and incentives in place.

5 Future Directions

The revised Australian KM Standard (AS 5037—2005) recognises that an emergent KM generation requires radical changes to traditional organisational cultures to create environments where knowledge workers are authorised, empowered and encouraged to cooperatively manage their own work practices and knowledge.

From the onset, the organisation needs to appoint a core team of knowledge workers or “Wiki evangelists”, who are good writers and experts in their specialised fields who can come to a consensus on what the encyclopaedia should look like and "seed" the corporate Wiki. Wiki scribes can help those who are not comfortable with technology or are not fluent writers. The adoption of an incremental principle points out to the non existence of pages which tempts users to create new pages of content e.g. produce an annual report or submit ideas for a group project, as part of the workload. As employees grow more confident, the corporate Wiki can harvest contributions about declarative knowledge (know-what) e.g. 'best practices', business procedures and rules; procedural knowledge (know how) e.g. stories, conversations and other context-rich knowledge, and conceptual knowledge (know why) e.g. principles and laws (Agarwal et al. 1997). If this is made easier using the corporate Wiki than without it, employees may take on board the benefits and readily move to other tasks.

A democratic culture of knowledge sharing reinforces the notion that knowledge workers' reputations are enhanced by participation in collaborative projects, acquire marketable job skills and knowledge; and increasing social recognition and prestige, just as people are rewarded for collaborative professional work. Traditionally, very few powerful people dominate the channels of information. The creation of the Internet has had a democratising effect on the availability and use of information. Things that seem to matter in the real world, such as age, social status, and level of education, are often dismissed as unimportant online. The same democratising effect will be true with Wikis. As it is not easy to transfer the cumulative experience and skills of employees to the organisation, a corporate Wiki can address this problem by being a 'peer production information commons' (Benkler, 2006). A corporate Wiki that is based on an open source model promoting a participatory and bottom up approach, can be common spaces where people share experiences and have unanticipated, un-chosen exposures to the ideas of other people.

Successful collaborations from the OSM such as Linux were created outside the business environment and have become mainstream. *Wikipedia*'s popularity is due to its open, free and collaborative nature that helps meet the challenges of a connected world. If corporate Wikis can borrow elements that contribute to Linux and *Wikipedia*'s success, while at the same time addressing their limitations, corporate Wikis will get an opportunity to prove how mainstream this new generation KMS can become.

References

Agarwal, R., Krudys, G. and Tanniru, M. 1997, Infusing Learning into the IS Organization. European Journal of Information Systems, **6**(1):25-40.

AS5037 [Int] 2003, Interim Australian Standard Knowledge Management. Standards Australia

AS5037 2005, Australian Standard Knowledge Management. Standards Australia.

Benkler, Y. 2006, The Wealth of Networks. (February 20, 2007) http://www.benkler.org/wealth_of_networks/

Burstein, F. and Linger, H. 2003, Supporting post-Fordist work practices: A KM framework for dynamic intelligent decision support, Journal of IT&P special issue on KM, **16**(3): 289-305.

Drucker, P. 1994, The Theory of Business. Harvard Business Review, September-October, pp. 95-104.

Emigh, W. and Herring, S. 2005, Collaborative authoring on the web: A genre analysis of online encyclopedias. Proceedings of the Hawaii International Conference on System Sciences.

Engeström Y. 1987, Learning by expanding: An activity-theoretical approach to developmental research. Helsinki: Orienta-Konsultit.

Giles, J. 2005, Internet encyclopaedias go head to head, *Nature,* Dec. 15, http://www.nature.com/nature/journal/v438/n7070/full/438900a.html.

Hart D. and Warne L. 2005, Comparing cultural and political perspectives of data, information and knowledge sharing in organizations, Journal of Knowledge Management

Handzic, M. and Hasan, H. 2003, The search for an integrated KM framework. In Hasan, H. & Handzic, M. (Eds). Australian Studies in Knowledge Management, UOW Press, pp. 3-34.

Hasan, H. 1999, Integrating IS and HCI using AT as a Philosophical and Theoretical Basis, Australian Journal of Information Systems, **6**(2):44-55.

Hasan, H. 2003a, An Activity-based Model of Collective Knowledge. In Proceedings of the 36th Hawaii International Conference on System Sciences, Big Island, Hawaii, USA.

Hasan, H. 2003b, Communities as Activity Systems and other such Frameworks. In Hasan, H., Verenikina, I. and Gould, E.(Eds) Information Systems and AT, Expanding the Horizon, UOW Press, 3, pp.74-95.

Hasan, H. and Pfaff C.C. 2006, The Wiki: a tool to support the activities of the knowledge worker. In Proceedings at the Transformational Tools for 21st Century (TT21) 2006 Conference. Central Queensland University, Rockhampton, Queensland.

Hasan, H. and Pfaff C.C. 2006, Emergent Conversational Technologies that are Democratising Information Systems in Organisations: the case of the corporate

Wiki. Proceedings at the ISF: Theory, Representation and Reality conference, ANU, Canberra.

Klint, P. and Verhoef, C. 2002, Enabling the creation of knowledge about software assets. Data and Knowledge Engineering, **41**:2-3, 141-158.

Kurzidim, M. 2004, Wissenswettstreit. Die kostenlose *Wikipedia* tritt gegen die Marktführer Encarta und Brockhaus an. (Knowledge competition: Free *Wikipedia* goes head to head with market leaders *Encarta* and *Brockhaus*). Oct. 4, pp.132–39.

Lam, W. and Chua, A. 2005, Knowledge management project abandonment: an exploratory examination of root causes, Communications of the Association of Information Systems, **16**: 723-743.

Leontiev, A.N. 1981, Problems of the Development of Mind. Moscow, Progress.

Leuf, B. and Cunningham, W. 2001, The Wiki Way, Quick Collaboration of the Web. Addison-Wesley.

Lih, A. 2004, *Wikipedia* as Participatory journalism: reliable sources? Metrics for evaluating collaborative media as a news resource. Proceedings of the Fifth International Symposium on Online Journalism, April 16-17, Austin, Texas.

Nadler, D.A. and Shaw, R.B. 1995, Change leadership: core competency for the 21st Century. In Nadler, D.A., Shaw, R.B. and Walton, A.E. (Eds.), Discontinuous Change: Leading Organizational Transformation. San Franscisco, CA: Jossey-Bass.

Nonaka, I. 1991, The knowledge-creating company. Harvard Business Review, Nov/Dec, pp.96-104.

Pfaff, C.C. and Hasan, H. 2006, Overcoming organisational resistance to using Wiki technology for Knowledge Management. In Proceedings of the 10th Pacific Asia Conference on Information Systems, Kuala Lumpur, Malaysia.

Rosenzweig, R. 2006, Can history be Open Source? *Wikipedia* and the future of the past. The Journal of American History, **93**(1):117-46.

Sanger, L. 2005, The early history of *Nupedia* and *Wikipedia*: a memoir. In DiBona, C., Stone, M. and Cooper, D., editors, Open Sources 2.0, O'Reilly Press: Sebastopol, CA.

Thomas, H. and Torsten, P. 2005, Supporting knowledge work with Knowledge Stance-Oriented Integrative Portals, Proceedings of the European Conference on IS, Regensburg, Germany.

Wagner, C. 2006, Breaking the knowledge acquisition bottleneck through conversational knowledge management. Information Resources Management Journal, **19**(1):70-83.

Waycott, J., Jones, A. and Scanlon, E. 2005, PDAs as lifelong learning tools: An AT based Analysis. Learning, Media and Technology, **30**(2):107-130.

Wei, C., Maust, B., Barrick, J., Cuddihy, E. and Spyridakis, J. H. 2005, Wikis for supporting Distributed Collaborative Writing. In Proceedings at the Society for Technical Communication, Seattle, pp. 204-209.

EMERGENT DECISION-MAKING PRACTICES IN FREE/LIBRE OPEN SOURCE SOFTWARE (FLOSS) DEVELOPMENT TEAMS

ROBERT HECKMAN, KEVIN CROWSTON, U. YELIZ ESERYEL, JAMES HOWISON, EILEEN ALLEN, QING LI

School of Information Studies,

Syracuse University 344 Hinds Hall Syracuse, NY 13244-4100 USA

WWW home page: http://floss.syr.edu

{rheckman, crowston, uyeserye, jhowison, eeallen, qli03}@syr.edu

Abstract: We seek to identify work practices that make Free/Libre Open Source Software (FLOSS) development teams effective. Particularly important to team effectiveness is decision making. In this paper, we report on an inductive qualitative analysis of 360 decision episodes of six FLOSS development teams. Our analysis revealed diversity in decision-making practices that seem to be related to differences in overall team characteristics and effectiveness.

Key words: Decision making practices; free/libre open source software development teams; team effectiveness; FLOSS; OSS

1. INTRODUCTION

In Free/Libre Open Source Software (FLOSS) teams, decision-making practices emerge from the interactions of the team members rather than from organizational context. Discontinuities among team members make such emergence and indeed any kind of consistent decision process harder to attain, yet effective teams seem to have developed productive ways of making decisions. Developers contribute from around the world, meet face-

Please use the following format when citing this chapter:

Heckman, R., Crowston, K., Eseryel, U.Y., Howison, J., Allen, E., Li, Q., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 71–84.

to-face infrequently (some not at all), and coordinate their activity primarily by means of information communication technologies (ICT) (Raymond, 1998a; Wayner, 2000). Since FLOSS teams are representative of self-organizing teams, because they have shared goals and a user base and members to satisfy, and are interdependent in terms of tasks and roles, our findings will have broader implications for understanding other technology supported self-organizing teams.

Our objectives for this paper are two-fold: First, to present a descriptive analysis of the range and evolution of decision-making practices of FLOSS teams based on longitudinal observation of 120 decision episodes that took place in 6 naturally occurring teams. We chose projects that are similar in market size potential and software development stage, that use similar tools, and belong to one of two software categories: Instant Messaging (IM) Clients and Enterprise Resource Planning (ERP) Systems. We present this description in the form of multiple case studies that compare and contrast decision-making practices between teams.

A second objective is to relate differences in team work practices to team effectiveness. Because we compare teams that differ in effectiveness but are similar in other ways, we provide suggestions for future research on the relationship between decision-making practices and team effectiveness. This comparison, between teams in two different software categories, also enables us to understand how software properties such as software complexity, target market, and team nature can affect the decision making process.

2. LITERATURE REVIEW

In this section, we briefly review literature relevant to our study of decision-making practices in FLOSS teams.

Dean and Sharfman (1996) suggest a close link between decision making processes and decision effectiveness. Guzzo and Salas (1995) suggest a close tie between effective decision making and overall team effectiveness and the importance of understanding the practices by which decisions are actually made in teams. In the information systems (IS) literature more particularly, there have been numerous studies of ICT support for group decision making (e.g., DeSanctis & Gallupe, 1987a; Fjermestad & Hiltz, 1998/1999; Turoff, Hiltz, Bahgat, & Rana, 1993). Huber et al. (1986) suggest that decisions that groups need to make are becoming increasingly more complex, and requires greater participations and a faster decision making process. DeSanctis and Gallupe (1987b) expect greater and more even participation to yield desirable effects for the group. High participation from a group allows pooling of more resources and promotes error checking, thus enabling better decisions (DeSanctis & Gallupe, 1987b; Hackman & Kaplan, 1974; Holloman & Hendrick, 1972). Small group research suggests that higher participation in group decisions increases the acceptance of these decisions and members' increased sense of responsibility for those decisions (Bedau, 1984; DeSanctis & Gallupe, 1987b; Hackman & Kaplan, 1974). The small group research literature also identifies that higher participation in group decisions increases the level of group cohesion and individuals' satisfaction with the group (e.g., Hare, 1976).

High participation in group decisions may potentially slow down the decision making process, resulting in dissatisfaction with the decision making process. Yet, slowing processes in the FLOSS environment may be less problematic given that FLOSS projects are protected from the type of competition that their for-profit counterparts face.

Many of the studies on decision making in the IS field have been design focused, offering important suggestions to improve the process and quality of team decisions. Studies of groups in action have tended to adopt experimental methods and focus on single episodes of decision making rather than on practices over the life of an intact team (though there are exceptions, such as (Eden & Ackermann, 2001)). Broadly speaking, there are few studies that examine the kinds of decision processes that emerge in intact self-organizing teams, how these practices evolve over time, and how they contribute to overall team effectiveness. These decision processes include, but aren't limited to, how the decision process gets initiated and concluded, the types and roles of participants, and the frequency and quality of participation.

One common concern in several studies of FLOSS teams' decision making, has been the style of participation. At one extreme is a style where decisions are primarily made by a few central participants, even a single individual, as in Linux, where Linus Torvalds originally made most of the decisions for the team (Moon & Sproull, 2000). Such a decision style has been characterized as a "benevolent dictatorship" (Raymond, 1998b). On the other extreme are teams with a decentralized communications structure and more consultative decision-making style. Some teams even settle decisions by voting (Fielding, 1999). Although participation in decision making by a few key people at the core versus the people at all levels has been described in these studies, the connection between participation style and team effectiveness isn't clear. In addition, the participation in decision making might evolve over time as the project evolves. Fitzgerald (2006) suggests that a small group will control decision making early in the life of a project, but as the project grows, more developers will be involved. German (2003) documents such a transition in the case of the Gnome project. Thus, not only the extent and frequency of participation, but also the evolution of decision participation over time may influence the relationship between decision making practices and team effectiveness.

3. METHOD

To analyze decision-making practices in open-source projects, our research employs a multiple case study methodology, focused primarily on content analysis of decision-making discussions. To find these discussions, we analyzed the email discourse between administrators, developers, and users that takes place on the developers' e-mail lists or forums, which are the primary communication venue for the teams. Archives of these lists are available on project websites and from repositories such as SourceForge.net[1].

3.1 Case selection

We chose six FLOSS projects by considering several dimensions to balance maximization of variability and control of unwanted systematic variance. First, we controlled for topic. Projects within a single topic category are potential competitors, making comparisons of outcomes such as downloads between these projects valid. On the other hand, we wanted to have projects at different levels of complexity to provide for variability. Accordingly we picked three projects that develop Enterprise Resource Planning (ERP) systems (Compiere, WebERP and Apache OFBiz) and three teams that develop Instant Messenger (IM) clients (Gaim, aMSN and Fire). ERP projects are more complex than IM projects since they have high software code interdependencies, and many external constraints such as accounting rules and legal reporting requirements. One, Compiere, originated as a closed-source project, offering an opportunity to examine the consequences of that history.

Second, to minimize unwanted variance, we chose projects that are roughly similar in age and status (production/stable.) Projects at this stage have relatively developed membership and sufficient team history, yet the software code still has room for improvement, which enables us to observe rich team interaction processes. Third, the projects we chose varied in

[1] Because postings to lists are intended to be publicly accessible, our human subjects review board considers them public behavior, and so does not require formal consent to study them.

effectiveness. Project effectiveness is a multi-dimensional construct, including success of the project's outputs, team member satisfaction and continued project activities (Hackman, 1987). We therefore applied the multivariate approach to effectiveness in the FLOSS context suggested by Crowston et al. (2006) aiming to discover a rank order within the IM and

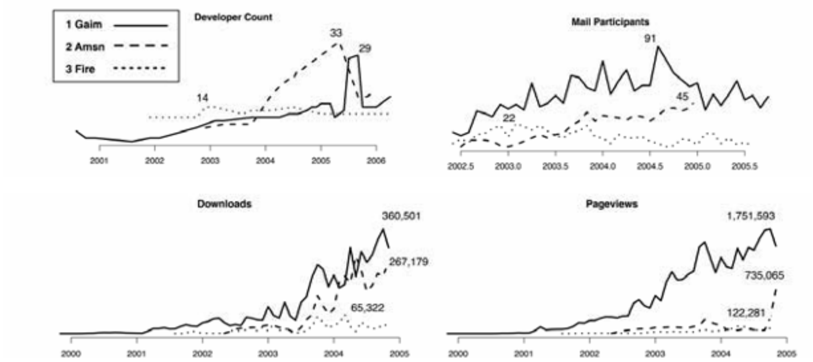


Fig. 1. Comparison of Effectiveness Measures for IM Projects

ERP categories. Project outputs were measured by downloads and page views, developer satisfaction was measured through development numbers and participation on the developer mailing lists. The array of measures presented in Figures 1 and 2 use data collected by the FLOSSmole project (Howison, Conclin, & Crowston, 2006) from the project establishment in SourceForge until around March 2006. According to analyses shown in Figures 1 and 2, the most effective IM project is Gaim, followed by aMSN then Fire, and the most effective ERP project is Compiere followed by OFBiz then WebERP.

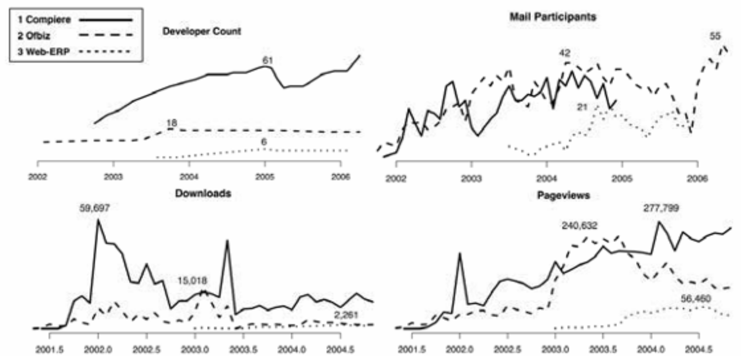


Figure 2. Comparison of Effectiveness Measures for ERP Projects

3.2 Unit of Analysis: Decision Episodes

We selected the decision episode as our primary unit of analysis. We define a decision episode as a sequence of e-mail messages that begins with a triggering message presenting an opportunity for choice (such as a feature request or a report of a software bug), includes discussion related to the issue, and an announcement of a decision about the opportunity.

We differentiated between decision episodes that focus on software code-related day-to-day decisions and those that focus on long-term strategic decisions (such as membership, infrastructure and marketing decisions). In keeping with our desire to focus on likely similarities to other forms of distributed teams, this paper focuses on the software code-related episodes.

In order to observe potential changes in decision-making processes and norms over time, we sampled 20 decision episodes from three comparable time periods in each project's life. For each project, the beginning and the ending periods are the first and last 20 decision episodes observable on the developer mailing list by May 2006. The middle period for each project consisted of 20 episodes surrounding a major software release approximately halfway between the beginning and ending periods. Figure 3 shows the specific time periods sampled for each project. Note that the sample periods differ in length due to different rates of development in the projects.

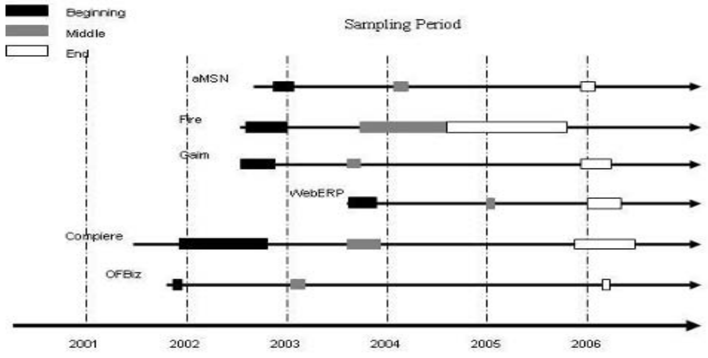


Fig. 3. Sampling Periods of IM and ERP Projects

3.3 Analysis and Coding of Episodes

We began analysis of decision episodes by coding observable, manifest elements of content that are directly related to the decision-making typologies described in the literature above. We coded: number of messages per episode, duration of the episode (in days), total number of participants in the episode, and the role of each message's sender: project administrator, developer (if listed developer according to the project webpage on SourceForge) or non-developer (if not listed on SourceForge). We considered administrators and developers "core" members of the team, and non-developers "peripheral" members.

Subsequent coding was inductive, with three independent analysts reading the episodes in order to understand the salient features of the decision process. Through several iterations, at least two independent coders identified and agreed upon four additional latent variables that were important to the decision process. Each episode was then formally coded by at least two analysts, with all disagreements discussed and reconciled to achieve essentially complete agreement. The latent variables identified and coded are decision trigger type, decision process complexity, decision announcement, and decision type.

3.3.1 Decision Trigger Type

One goal of our inductive content analysis was to understand the types of triggers that presented decision opportunities for the group. Thus, we developed a typology of triggers. Decision episodes about code are triggered by: (1) bug reports, (2) feature requests, (3) problem reports, which are different from bug reports since they may also include problems that end-users may be facing due to hardware, software or process reasons, (4) patch submissions (5) to-do lists and (6) mixed triggers that include one or more different trigger types.

3.3.2 Decision Process Complexity

Inductive analysis also indicated that some episodes required more complex decision paths than others. For example, some episodes involve a single choice that responds to a single straightforward trigger. These were coded as "Single." Others responded in a linear, straightforward fashion to a trigger that contained multiple opportunities for choice (e.g., a release to-do list). These were coded as "Multiple-Simple." The most complex episodes were not straightforward or linear in nature. Regardless of the nature of the initial trigger, in these episodes new, sometimes unrelated triggers created additional opportunities for choice. The initial problem(s) might be solved or not, and the new problems introduced might also remain unsolved. These episodes, coded "Multiple-Complex," closely resemble the garbage can decision opportunities described by Cohen et al. (1972) in that a straightforward, sequential "problem-resolution" decision process was not observed.

3.3.3 Decision Announcement

In order to reliably determine that a decision had truly been reached, our independent coders coded the statement(s) that confirmed that a decision had been reached.

3.3.4 Decision Type

Our analysis also coded when the main decision announcements reflected either acceptance or rejection of a need for code change, acceptance or rejection of the suggested code change, or both.

This initial typology of latent variables provides the ability to concisely describe multiple characteristics of the decision making process, and allows us to measure the participation of various members in decision making, thus contributing to our first objective, providing a rich description of the evolution of decision-making practices over time and the connection between decision making and FLOSS effectiveness.

4. FINDINGS

Our research objectives were to present a descriptive analysis of the range and evolution of decision-making participation in FLOSS teams, and to relate differences in these work practices to team effectiveness. In order to do that we present below differences in decision episode participation between more and less effective teams. We begin by first discussing overall participation in decision episodes. We then discuss relative participation by core and peripheral members, and finally, present an analysis of who triggers decision episodes and who announces decisions.

4.1 Participation in Decision Episodes

The overall number of participants in episodes increases from the first to last period for effective projects such as Gaim, aMSN and OFBiz (see Figure 4). Fire, the least-effective IM project, did not see an increase in the average number of participants per episode. Similarly, WebERP, the least effective of the 3 ERP projects, increased its average number of participants in the middle period, but did not maintain the participation. Compiere, despite its apparently high effectiveness, exhibits a different participation behavior than the other projects, perhaps due to a project management style that remains from its closed-source days.

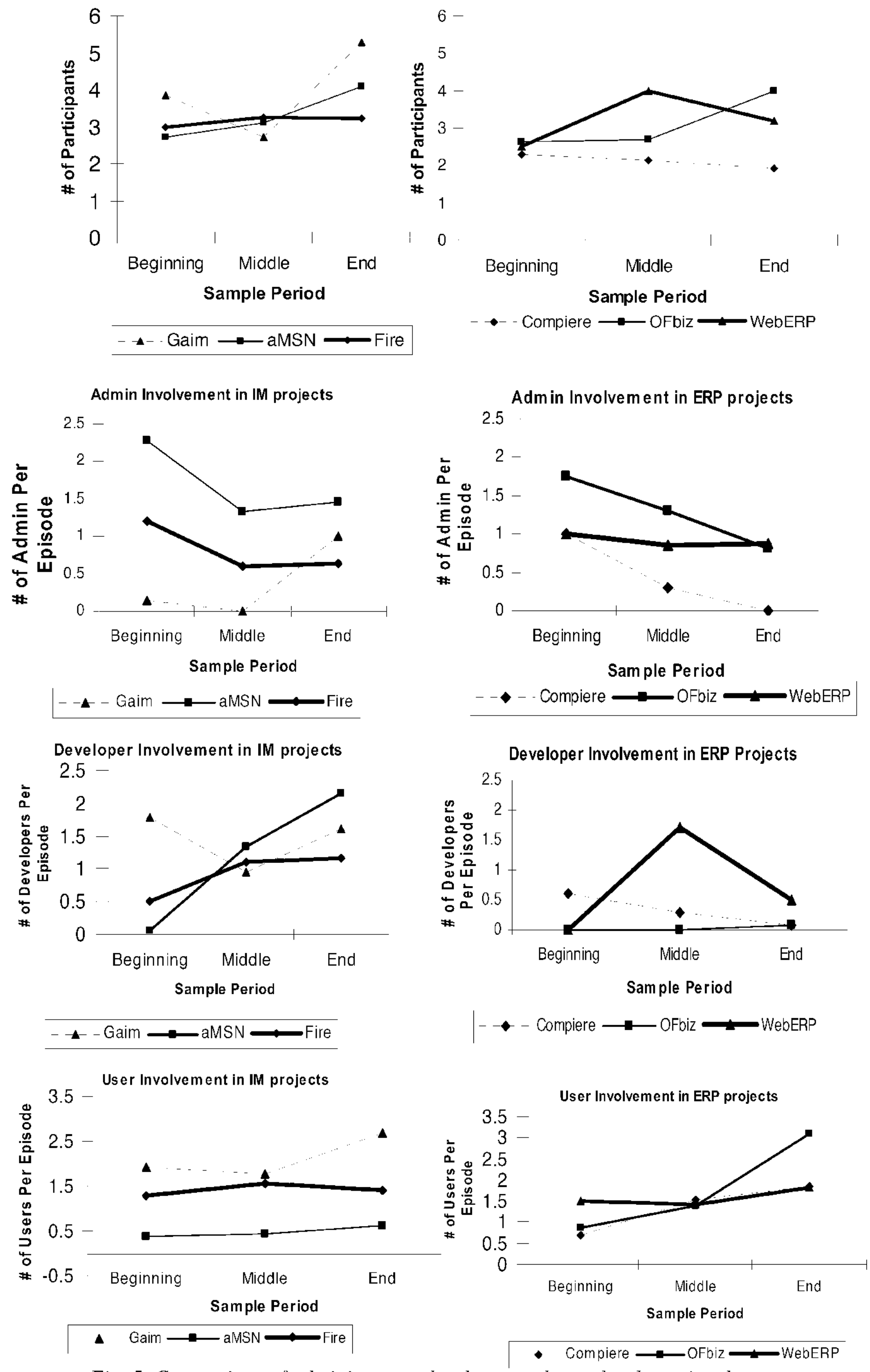


Fig. 5. Comparison of administrator, developer and non-developer involvement

When participation is analyzed in detail, we see that in most projects administrators were highly involved in the initial phase, perhaps to set up the project, communicate to the community and attract project team members (Figure 5). In later periods, the administrators' involvement diminished, allowing the development of a self-governing community. Because of a leadership change, Gaim is an exception to this pattern . In the beginning period, an administrator who was phasing himself out of the project remained relatively silent, allowing developers to actively participate in decision making. Between the middle and end period an active developer became the administrator and remained an active participant in the discussions.

As the administrators became less involved in discussions, the periphery became more involved. However, we see a difference in involvement across project types. In IM projects the developers show a clear pattern of taking more charge in discussions, especially for more effective projects (except for the dip in the middle period for Gaim due to the leadership change). Also, all IM projects show an increase in the non-developer involvement between the first and last periods. The least effective IM project Fire shows an increase in both developer and non-developer involvement for the middle period, but does not sustain this trend towards the end. In the ERP projects, patterns in developer involvement are less clear, however, non-developers are increasingly involved in decisions over time, at an even faster rate than in the IM projects.

These comparisons suggest that although there are similarities across IM and ERP projects, there are some differences in how decision participation evolved over the sampling intervals. Over time, non-developer participation in decision episodes increased for effective projects and administration participation decreased for almost all projects, yet the non-developer and developer participation showed different patterns based on the project type.

4.2 Who Triggers Decision Episodes and Announces Decisions?

In order to better understand the role played by core and peripheral members of the projects in creating decision opportunities for the group, we examined who sent the e-mail message that triggered decision episodes, and who sent the message(s) that announced decisions. Figure 9 shows that both core (administrators plus developers) and periphery played a role in triggering decision opportunities. Figure 6 shows both the IM and the ERP projects from the most effective to the least effective within their software categories. In more effective ERP projects, decision episodes are triggered by the periphery, whereas IM projects show no specific trend. Across all projects, core members initially create opportunities for decision (i.e., triggers) and in time, this activity moves to the periphery.

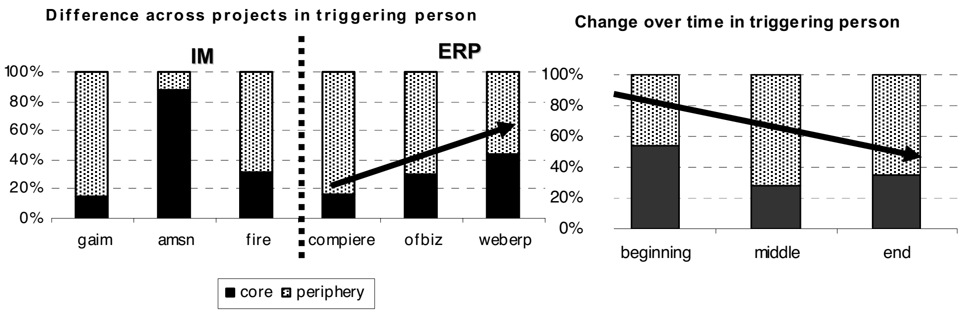


Fig. 6. Comparison of the involvement of the core and periphery in triggering decisions

Figure 7 shows that both core and periphery played a role in announcing decisions. Although across all projects, the members from the core project team announced more decisions, the most effective projects within their categories, i.e., Gaim and Compiere, exhibited higher peripheral involvement in decision announcement than the others. This difference between projects was statistically significant (X^2 =22.038; df=5; p<.01). Across all projects, although most of the decisions were announced by the core, over time, peripheral involvement increased. This difference in peripheral involvement over time was also significant (X^2 =16.204; df=2; p<.01) .

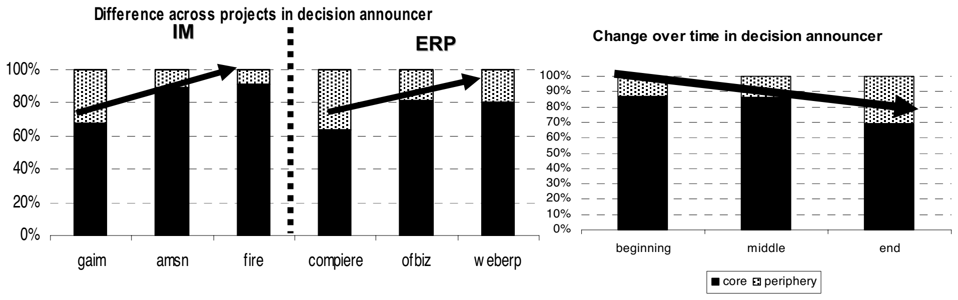


Fig. 7. Comparison of the involvement of the core (administrators and developers) and periphery (users) in announcing decisions

5. DISCUSSION AND CONCLUSION

We presented findings from a long-term research project that seeks to identify work practices that make FLOSS teams effective. This paper described participation in code-related decision-making practices in six open source project teams. In general we found evidence to support the expectation of the literature that greater participation in decision making would be associated with more effective projects.

Generally, among the more effective projects, the number of participants in decision making episodes increased over time, whereas less effective projects either showed a reducing trend or did not sustain initially increasing participation. Similarly, effective projects showed high administrator involvement in decision episodes at the beginning phase of the projects, followed by declining administrator participation coupled with increasing developer and non-developer participation in later periods. (Gaim, although the most effective project in its category, appears to be an exception to this pattern, with low administrator involvement in the first two periods. Gaims pattern is understandable, however, when we recognize that there was a change in administrators during this period. Gaim has very high developer involvement in period 1 may show developers attempting to fill this vacuum.) High administrator involvement in decision making in early phases of these projects may indicate an attempt to establish decision-making standards and norms, while declining involvement may signify increasingly empowered developers and non-developers. Interestingly, less effective projects (Fire, WebERP) show a similar decline in administrator participation in decision making over time, but without a corresponding increase in non-developer involvement.

We also observed higher decision-making participation by the periphery (non-developers) in more effective projects. With one exception (aMSN), a higher percentage of decision opportunities were triggered by non-developers in more effective projects (Figure 9). Similarly, the more effective projects had a higher percentage of non-developers making decision announcements, (Figure 6) even though the core made a higher percentage of decision announcements overall. These two findings–higher overall participation and higher participation by the periphery in more effective projects–supports the notion that increased participation and diversity in decision-making practices is related to improved team performance.

Our findings also suggest that while there are common trends, there are interesting differences between projects in decision-making practices. For example, Compiere showed differences from other projects almost on all measures. This may be because previously it was a closed-source proprietary company and that the project was not originally self-organizing or emergent. Another such difference can be observed between IM and ERP project categories in the evolution of developer and non-developer participation over time. In IM projects developer participation in decisions grew rapidly, while in ERP projects, this did not occur (Figure 5). Non-developer participation grew more rapidly in ERP projects than in IM projects. This may be explainable by differences in the software type, which in turn affects the type of end-users. IM clients are typically used by individuals, who seek solutions to the issues they face or enhancements to the software for their own needs. On the other hand, the end-users of ERP software are companies, whose own in-house developers play the role of end-users in the FLOSS projects. These end-users are likely to have a high professional interest in the software, and thus high and sustained involvement throughout the project.

By presenting this comparative analysis of the range and evolution of decision-making practices we have begun the process of relating differences in work practices to team effectiveness. These findings suggest that more participative and diverse decision making practices are positively related to team effectiveness in technology supported self organizing teams. The findings also reinforce the idea that all open source teams are not alike. Differences in contextual attributes such as software type and function, as exemplified by the comparison of IM and ERP projects, have an influence on the emergence of work practices. We believe that the variables and relationships we have identified provide the foundation for deeper exploration and potentially richer explanations of the relationships we have described. Future studies should replicate and extend this analysis to additional FLOSS projects, and to other technology supported self organizing teams. A useful future study would be to analyze the decision making process to identify various decision styles by various teams and the relationship between decision styles and team effectiveness.

6. REFERENCES

Bedau, H. (1984). Ethical aspects of group decision making. In W. C. S. a. Associates (Ed.), *Group Decision Making* (pp. 15-150). Beverly Hills: Sage Publications.

Cohen, M. D., March, J. G., & Olsen, J. P. (1972). A garbage can model of organizational choice. *Administrative Science Quarterly, 17*, 1–25.

Dean Jr, J. W., & Sharfman, M. P. (1996). Does decision process matter? A study of strategic decision-making effectiveness (Vol. 39, pp. 368-396): JSTOR.

DeSanctis, G., & Gallupe, B. (1987a). A foundation for the study of group decision support systems. *33*(5), 589–609.

DeSanctis, G., & Gallupe, R. B. (1987b). A Foundation for the Study of Group Decision Support Systems. *Management Science, 33*(5), 589-609.

Eden, C., & Ackermann, F. (2001). Group decision and negotiation in strategy making. *Group Decision and Negotiation, 10*(2), 119-140.

Fielding, R. T. (1999). Shared leadership in the Apache project. *Communications of the ACM, 42*(4), 42–43.

Fitzgerald, B. (2006). The transformation of Open Source Software. *MIS Quarterly, 30*(4).

Fjermestad, J., & Hiltz, S. R. (1998/1999). An assessment of group support systems experiment research: Methodology and results. *Journal of Management Information Systems, 15*(3), 7–149.

German, D. M. (2003). The GNOME project: A case study of open source, global software development. *Software Process: Improvement and Practice, 8*(4), 201–215.

Guzzo, R. A., & Salas, E. (1995). *Team Effectiveness and Decision Making in Organizations*. San Francisco: Jossey-Bass.

Hackman, J. R., & Kaplan, R. E. (1974). Interventions into group process: An approach to improving the effectiveness of groups. *Management Science, 5*(3), 459-480.

Hare, A. P. (1976). *Handbook of small group research* (2nd ed.). New York: Free Press

Holloman, C. R., & Hendrick, H. W. (1972). Adequacy of Group Decisions as a Function of the Decision-Making Process. *The Academy of Management Journal, 15*(2), 175-184.

Howison, J., Conclin, M., & Crowston, K. (2006). FLOSSmole: A collaborative repository for FLOSS research data and analysis. *International Journal of Information Technology and Web Engineering, 1*(3), 17-26.

Huber, G. P., & McDaniel, R. R. (1986). The Decision-Making Paradigm of Organizational Design. *Management Science, 32*(5), 572-589.

Moon, J. Y., & Sproull, L. (2000). Essence of distributed work: The case of Linux kernel. *First Monday, 5*(11).

Raymond, E. S. (1998a). The cathedral and the bazaar. *First Monday, 3*(3).

Raymond, E. S. (1998b). Homesteading the noosphere. *First Monday, 3*(10).

Turoff, M., Hiltz, S. R., Bahgat, A. N. F., & Rana, A. R. (1993). Distributed group support systems. *MIS Quarterly, 17*(4), 399–417.

Wayner, P. (2000). *Free For All:How Linux and the Free Software Movement Undercut the High-Tech Titans*. New York: HarperCollins.

Experiences on Product Development with Open Source Software

Ari Jaaksi
Nokia
P.O Box 779
33101 Tampere, Finland
ari.jaaksi@nokia.com

Abstract. This article discusses Nokia's experiences of using open source in commercial product development. It presents the development model used in the creation of mobile consumer devices and highlights the opportunities and challenges experienced. This article concludes that the main benefits come from the utilization of already available open source components, and from their quality and flexibility. It illustrates the challenges and solutions faced when mixing open and closed development models at Nokia.

1 Introduction

The Nokia 770 and N800 Internet Tablets are mobile consumer devices. They provide wireless internet access and enable internet use cases such as voice and video calls, web browsing, messaging, and media consumption in a pocketable mobile device. Nokia has built these products on Linux and other open source components in a close collaboration with open source communities. In addition, Nokia runs the www.maemo.org web site that supports community development on internet tablets.

Nokia uses open source extensively in the creation of the internet tablets. We favor components that are developed by active communities and used by many users. This ensures that the selected components are developed and maintained properly both now and also in the future. For this reason we prefer to use mainstream desktop components whenever possible. Desktop and PC related projects are typically more active and mature than the projects targeting embedded devices. Therefore, we actually run a Linux based desktop configuration on a mobile device.

2 Software Architecture

Figure 1 illustrates our software architecture [1]. We integrate unmodified open source components into our platform. We also sponsor the enhancements of many existing open source components to make them fit for our use. We then integrate

Please use the following format when citing this chapter:

Jaaksi, A., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 85–96.

these modified open source components into our platform. Some components we develop from scratch, many of which we then open source. Finally, we integrate closed components from various sources, such as from commercial software vendors.

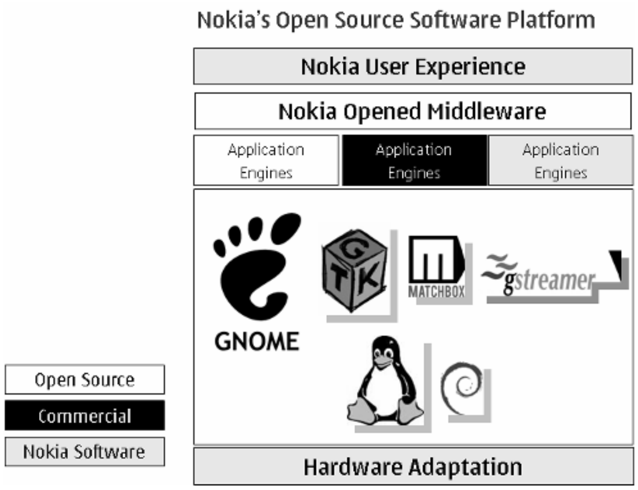


Figure 1: The Nokia open source software architecture

We have 428 source code packets in our platform. 25% of the packages are taken from open source projects without any modifications. Examples of such components include the gnuchess chess game engine, bzip2 data compressor, and id3lib for manipulating ID3v1 and ID3v2 tags in digital audio files.

About 50% of the packages originate form open source projects, but Nokia has made modifications to the components. In such cases, we actively push our modification upstream to the originating projects. The additional modification work is needed especially in the areas of UI and usability, power management, performance, and memory management. Our engineers work directly with communities participating development projects to ensure that our modifications are accepted upstream. In addition to making modifications ourselves, we also hire and ask developers within the communities to enhance components based on our needs. Examples of such components are the Linux kernel, D-BUS, GNOME-VFS, GTK+, GStreamer, and OBEX. We also reuse and improve entire subsystems and subsystem architectures, such as GNOME [2] and Debian [3]. Instead of separate components, we then reuse architectural blocks that already integrate several independent components.

Finally, some 25% of the packages are proprietary closed source components, either belonging to Nokia or licensed from commercial vendors such as Real Networks or Adobe. Some of the Nokia proprietary components that are kept closed are closely related to the hardware. Examples of such components are the boot loader and battery charging implementations. In addition, the majority of the user interface applications are also closed.

A European Union report by Ghosh [4] studied the software running on the Nokia 770. They concluded that the device runs 15 million lines of open source code, 200.000 of which where created by Nokia. This demonstrates that it is possible to use open source code and modify it to meet your own specific needs with minimal effort. In fact, Nokia manages to modify and use open source components for desktop environments in its mobile internet devices with less than 1.5% additional investment.

3 Community collaboration

We source our open source components directly from community projects. We do not use any embedded distros as the starting point of our architecture. Instead, we want to utilize mainstream desktop oriented open source components to get the maximum community benefits.

3.1 Selecting the core components

We analyze the technical suitability of all components and subsystems. All selected components must fulfill our functional requirements and meet our hardware specifications. The components also need to be of good quality and mature enough for consumer products.

We actively participate in the communities from which we source our components to ensure that the selected subsystems develop further over time. We also ensure that the goals of the development communities match our goals. This all happens through active community discussions, conferences, and workshops.

It is important that the open source components we use are licensed under proper licenses and have clear copyright and licensing information attached to them. We also choose to select components that do not lock us into one vendor through requirements such as mandatory copyright donations or dual licensing models. It is also important that our components be licensed under an open source license, such as LGPL, that allows us to integrate proprietary components into our platform as well.

For the key components and subsystems, we did not have too many options to choose from. For example, the only true graphical environment alternatives were Qt and Gtk+ [5][6]. We selected Gtk+ because it is developed by a vibrant multi-polar community with no single company dominance. It is therefore easy to contribute our changes to Gtk+ on the basis of general usefulness and technical merit only. Also, Gtk+ is licensed under LGPL, and that allows us to mix proprietary UI elements without a dual commercial license.

3.2 Creating software as a part of communities

Our strategy is to find a suitable community and then take part of the community work. We do not want to control the project or branch the work. Instead, working as an integral part of a community provides us with access to code and engineers outside of our own development team.

As an example, we sponsored the development of the D-BUS [7] message bus system. We hired some key developers from the D-BUS community into our project but asked them to continue working within the project in open source. They then contributed code and participated in the development of D-BUS, and we performed a lot of testing that helped in reaching the needed product quality.

We open source new components and subsystems we have developed, such as our Hildon application framework. We have also opened the development of selected middleware components at the Maemo Sardine distro [8]. Open middleware development enables application developers to follow the latest changes in our code so they can test their applications against the latest changes, update them as a result of any API changes, and pilot the latest additions to our software. This open development allows anybody to participate in the development of the middleware code and see where it's heading. This is all available before a stable release of the software for the end-users. As an example, several parts of the code running on N800 Internet Tablet were already available before Nokia even announced the product in early January, 2007.

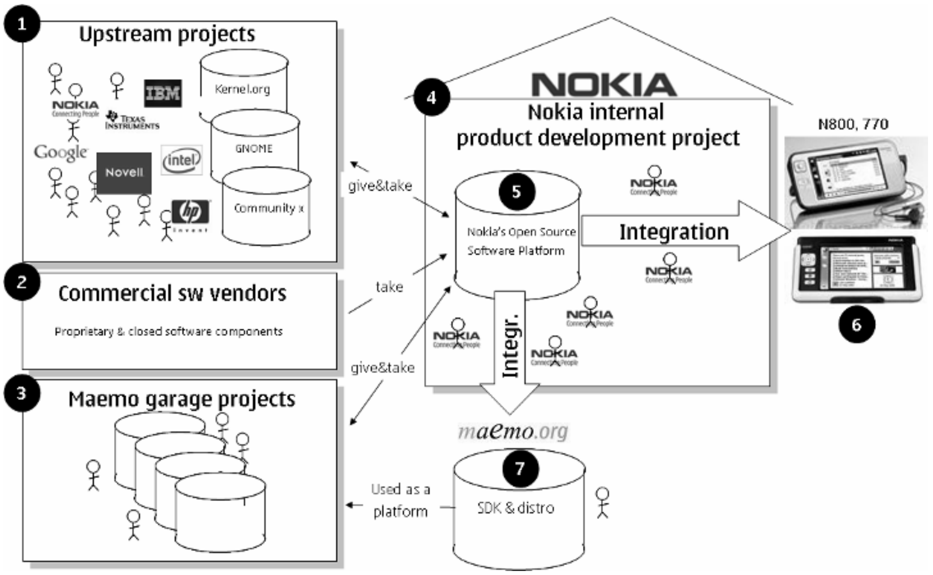


Figure 2: Working with communities

We work closely within communities to develop software, as illustrated in Figure 2. We collaborate with many individuals and companies in upstream projects (1) and Nokia engineers take part of the community work. We take selected components from those upstream projects (1), develop some code of our own (4), source components from commercial vendors (2) and create a Nokia internal distro called

Nokia's Open Source Software Platform (5). We then actively push our changes and modifications back upstream to minimize Nokia specific code.

We integrate the final software for our products within Nokia (4). We integrate both the product software for internet tablets (6), and the www.maemo.org tools and software for an external www.maemo.org software distro (7). While open source and communities help us in implementing software components and subsystems upstream (1)(3), we believe that the final product and product integration is better done within Nokia by Nokia (4). After all, we are responsible for meeting our quality, schedule, and monetary goals.

Finally, we offer www.maemo.org and Maemo garage for external developers (3). They use these facilities in their projects that develop software for Nokia's internet tablets. We are fortunate to have many volunteers and community people developing applications and submitting their work, such as documents, bug reports, and enhancements to www.maemo.org.

While Nokia provides the basic www.maemo.org infrastructure, the actual distro, and various development and community tools, it is the community members themselves who enhance them and provide support for each other. This greatly improves the developer experience on www.maemo.org. In the end of 2006, the www.maemo.org developer site hosted almost two hundred open source projects dedicated to the internet tablets, and had almost 60 000 unique visitors per month.

4 Benefits of open source

We have created two devices, provided software upgrades to these devices, and created an open source community around the www.maemo.org community site. Our development experiences include all the phases starting from initial requirements analysis to selling the devices, working together with many other companies and open source projects, and offering upgrade software for end users. We thus believe that we can draw some conclusions about developing consumer products with open source. The benefits are clear.

4.1 Efficiency

The biggest efficiency gains came from the utilization of already available components, such as the Linux kernel and the GTK+ toolkit. It was cheaper and more efficient for Nokia to build the internet tablets using the open source model than it would have been using a proprietary one. This conclusion can be drawn by studying other similar product development activities at Nokia.

In reality, developing an own operating system and middleware was never even an option for us. We needed to either use an existing commercial and closed operating system and middleware, or then use an existing open source operating system and middleware. We used the open approach in order to benefit from the cheaper or non-existent licensing costs, in order to have better strategical control, and to have the ability to freely enhance the code according to product and market needs.

Productive software developers can enhance development efficiency significantly. With proprietary and closed software systems, we typically train and educate developers for a long period of time. It takes several projects for the developers to become productive with the closed and proprietary systems and technologies we use at Nokia. This is not the case with our open source based software platform, because we use widely known tools, components and architectures. The Linux operating system and Debian packages, for example, are commonly taught in universities and other companies. That makes new developers productive faster than with other software platforms used at Nokia.

4.2 Quality

The code that we obtain from open source projects is of better quality and has fewer errors than code we developed by ourselves. This is because open source code has already been used by others before we take it into use, and they have already fixed the most severe errors.

However, if we compare the open source code to the commercial components used in our platform, the quality difference is not that obvious. The commercial components have typically been used by others, too. That has improved their quality.

An additional benefit to open source is that the quality of the code and the skills of developers can be verified in advance. We can study the component code, build prototypes, and run performance tests freely with open source components. This helps us to select good quality components and subsystems. Also, when a developer or a subcontractor submits code to an open source project, the quality is easy to verify. This allows us to assess the quality of our developers and subcontractors before hiring them.

4.3 Flexibility

Open source provides flexibility when we need to fix problems or change functionality. We often request bug fixes or modifications for the commercial closed components on our platform. However, if the vendor of that particular component does not have the capacity or willingness to fix the problem on time, we can be left few options. Typically we cannot fix problems ourselves in these scenarios, because the companies from whom we license our closed components don't want us to access their source code. With open source components, however, we fixed bugs ourselves, hire somebody else to fix them, or work with the communities in order to obtain the modifications. With so many options available, we are able to fix the problems we have in most cases.

4.4 Software licensing

Software in-licensing requires a lot of negotiations between a licenser and a licensee. Based on our experiences, an average in-licensing process for a software component takes 6 – 12 months. Problems and delays in software licensing are one of the most common reasons for missing features or delayed projects.

In contrast, licensing with open source is simple. The licensor already has the license terms in place. She may offer some additional options, such as support or training, but the actual licensing terms are already established. In addition, all the source code is available for the licensee to study and evaluate. The licensee can also assess the community, companies, and available hackers supporting the technology in question prior to taking it into use. And, they are able to talk to others about the technology without worrying about trade secrets. Because of these factors, open source projects are never delayed because of complicated in-licensing negotiations.

Open source simplifies and accelerates software licensing, and reduces technology and quality risks. Instead of negotiation for months, the technical work can start immediately.

4.5 Future and roadmaps

The future direction and plans for open sourced components and subsystems are typically discussed openly, and are open for contributions. We can, therefore, monitor and influence the development of relevant technologies through the community work.

The choices are more limited with closed source commercial components. Companies developing closed source components typically decide themselves about the future of their technology. They may choose to reveal parts of they plans, and they may choose to take external input into account. But, unlike in the open source, you cannot participate yourself and contribute in an open fashion.

4.6 Open source and confidentiality

An open source approach requires openness and information sharing during development. However, you do not want to reveal the products to the public before the actual product announcement. There is thus a potential conflict between the open source openness and product launch secrecy.

Nevertheless, we worked intensively with communities already before we announced the Nokia 770 Internet Tablet. Also, we opened parts of the firmware development before launching the Nokia N800; we worked with several communities to develop code for it. Many community developers had very detailed information about our forthcoming products due to their involvement in this process. Despite this, however, we had no information leakage from developers prior to the commercial product announcements. Based on our experiences, therefore, it is possible to develop software openly, while maintaining product confidentiality.

5 Issues and challenges

The open source development model is different than the closed one. The open source model shares code and work with other people and companies. They have their own schedules and targets that may not coincide with ours. However, that doesn't necessarily matter as long as we work upstream on non-differentiating aspects of the software.

At the end of the day, however, we must ensure that we get our products done in time with proper quality. Thus, at some point we need to drive the project to the conclusion that benefits us the most. This typically requires a more closed and single company controlled way of working. This mixture of open and closed development, illustrated in Figure 2, is important to master and we already can draw some conclusions from our experiences.

5.1 Hacking vs. stabilizing

In the early stages of a product development project, we work closely with communities, individual hackers, and hacker companies. We develop code in a true hacking mode. Later, we freeze our requirements to get things focused and to get software ready for shipment.

We have an internal milestone when we all software functionality must be implemented. We predict the shipment date to synchronize marketing activities, and reserve a factory production line. At this milestone, System Testing can run all test cases. All features are implemented at this point, but the system is still unstable and buggy. From this point on, all effort is put into bug fixing and stabilization.

At this milestone, the whole development team switches modes, from hacking and new development, to integration and stabilizing. Hacking and new development happens around independent components within teams. Integration and stabilizing, on the other hand, happens around the entire software stack and between teams. This requires a shift from a component view to a system view of software development.

This is a radical change of mode. The open source culture is very much for trials, hacking, innovation and other creative aspects of software development. Meeting deadlines, not developing new features, and focusing on stability are not what many open source communities or developers naturally do. In addition to us, the Linux project, Debian, and others seem to have difficulties making a final good quality release on time [9], [10].

In recent projects we have made the move from the hacking to stabilizing more apparent and strict. We now make the change very explicit in our process, and enforce it even more than in some conventional product development projects. Accordingly, we managed to get N800 ready right on time with no delays. This proves that we have managed to improve our stabilization phase.

5.2 Architecture management

Open source requires us to manage our architecture not only form the conventional 4+1 point of view [11] but also from the legal and IPR point of view. Some components, such as players and codecs, are available only as closed source components. We need to, therefore, mix open source and proprietary code and manage different licensing rules within the product code.

We manage the legal and IPR status of each software component. It is not enough to manage the architecture in terms of the development time API compatibility or run time performance, but we also need to manage the mix of various open source and closed source licensing rules. This is an additional job that we need to do when developing products based on open source.

5.3 Community alignment versus backwards compatibility

Internet tablets provide a platform on top of which applications and services can be developed. It is important that the platform provides application compatibility over product generations.

A binary compatibility is an ultimate goal for such backwards compatibility. That would allow the same application to run on various platform and product generations without recompilation. This is typically achieved by selecting development APIs and components that remain unchanged over platform generations. If a company developing product generations also develops all the software, such API freezing can be achieved by not introducing new features and changes to the relevant code.

However, open source components develop constantly. They get new features, their architecture change, and bugs are fixed. This development happens and it cannot be stopped. Our current strategy is to stay close to the latest development. We want to avoid a big difference between the component developed in the community and the component version used by us. For example, we are eager to move to the latest Linux kernel version as soon as it is possible for two reasons. We want to get the latest development into use, and we want to minimize the delta we need to maintain our selves.

These two concepts, product backwards compatibility and staying current with open source development are somewhat contradictory. In several cases, it would have been easier for us to provide backwards compatibility simply by not changing the underlying code. However, communities move on. If we decided to use an old version of a component we would need to do all backporting and new development ourselves to that component. Communities would not help us for they are already working on new versions. That would eventually put us on a different development branch and increase the amount of code we need to develop and maintain ourselves.

We do not have a final answer to this contradiction. So far we have not managed to maintain true backwards compatibility as we had hoped. Going forward, this is one of the things we need to better understand and manage. To do so, we are implementing more strict architecture management and compatibility layers between of the open source originated platform components and applications.

5.4 Community participation in product integration

We work closely within communities in the upstream projects to develop various software components. This work happens in a community mode. Then, when we decide on the product features and integrate the final software, we do it ourselves in Nokia internal product development projects, illustrated in Figure 2. The closed way of integrating the software causes frustration among some external open source developers. They'd like to find a way to be part of the Nokia's product development, not only in developing components and technologies in upstream projects but actually deciding on features, and integrating the final product together with Nokia.

We launched the Sardine distro [8] partially to address this problem. We now allow external developers to participate more in the actual development and integration process within Nokia. This is this is one of those areas where we still need to collect experiences and learn. There may be room for more community collaboration even in the most crucial steps of the product development. But in all cases, we expect that the product companies, such as Nokia, must have the final control over the product features and quality. It cannot be given to communities.

5.5 Investing in community work

Using open source code effectively requires community participation. It is sometimes possible to use an open source component without further development. Such participation is almost free of charge. In many cases, though, we work with communities to enhance components and develop them further. Such participation requires extra resources.

When open sourcing our own code and patches, we must ensure that our developers can work with the open sourced components. We must continue support the code in open source so that the code will meet our future needs, too. Just releasing code with no plans to develop it further won't benefit us.

We have open sourced individual components and participated some development with no clear benefit for us. We have either been left alone to develop the component, or our needs have not been taken into account when developing a component further. In these cases the joint open source development didn't happen or it didn't benefit us. Therefore, we now observe individual projects and try to identify when the open source investment pays off and when it doesn't.

6 Summary

We have created two consumer devices, the Nokia 770 and the Nokia N800 Internet Tablets, utilizing open source software. In addition, we have made software updates to those devices and initiated community work around the www.maemo.org web site. Our experiences demonstrate that an open source technology and development model is well suited for consumer devices. We have created products in a shorter time and with fewer resources with open source than we have managed to create using proprietary software alone. In essence, open source offers time and cost savings in a form of readily available components and subsystems, available developers, and an effective development model.

Open source doesn't make software development free or easy. It provides effective tools for product creation. Combining these new tools, such as community involvement, and utilization of open components, with more traditional software and product engineering practices is a good mix.

As a device manufacturer, we alone are responsible for the quality of the end product. We must therefore utilize all quality and software engineering mechanisms to achieve the needed quality. We cannot skip such development aspects such as specifications, integration, testing, and documentation, for example. In addition, open source introduces certain new requirements, such as community interaction and legal and IPR management. These hard requirements seem to contradict with open community work in certain cases. We have not managed to successfully solve all these conflicts. However, we are working on improving community participation in the stabilization process as well as allowing community members to participate firmware development. The results are not yet known, though.

Acknowledgements

Thanks to Bradley Mitchell for his assistance in writing this article.

References

[1] Jaaksi 2006: Building consumer products with open source
http://www.linuxdevices.com/articles/AT7621761066.html

[2] http://www.gnome.org/

[3] http://www.debian.org/

[4] Ghosh, R A (ed.), 'Study on the:Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU, Final report, November 20, 2006, Merit

[5] http://www.trolltech.com/

[6] http://www.gtk.org/

[7] http://www.freedesktop.org/wiki/Software/dbus

[8] http://sardine.garage.maemo.org/

[9] Glance, David, G : Release criteria for the Linux kernel, 2004:
http://www.firstmonday.org/issues/issue9_4/glance/index.html

[10] Brockmeier, Joe, The 2005 Debian Project Leader election, 2005 http://lwn.net/Articles/127031/

[11] Kruchten, P.B. The 4+1 View Model of Architecture. In IEEE Software, November, 1995: 42-50

Exploring the Effects of Coordination and Communication Tools on the Efficiency of Open Source Projects using Data Envelopment Analysis

Stefan Koch
Vienna University of Economics and Business Administration, Institute for Information Business
Augasse 2-6, A-1090 Vienna, Austria
stefan.koch@wu-wien.ac.at
WWW home page: http://wwwai.wu-wien.ac.at/~koch/

Abstract. In this paper, we propose to explore possible benefits of communication and coordination tools in open source projects using data envelopment analysis (DEA), a general method for efficiency comparisons. DEA offers several advantages: It is a non-parametric optimization method without any need for the user to define any relations between different factors or a production function, can account for economies or diseconwhile omies of scale, and is able to deal with multi-input, multi-output systems in which the factors have different scales. Using a data set of 30 open source project retrieved from SourceForge.net, we demonstrate the application of DEA, showing that the efficiency of the projects is in general relatively high. Regarding the effects of tool employment on the efficiency of projects, the results were surprising: Most of the possible tools, and overall usage, showed a negative relationship to efficiency.

Keywords. Open Source Software Development, Efficiency, Data Envelopment Analysis, Software Repositories

1 Introduction

Considerable uncertainty has for a long time surrounded the topic of the efficiency of open source software development, and the factors influencing this efficiency. Currently, any comparison of open source software projects is very difficult. There is increased discussion on how the success of open source projects can be defined [21,22,9,10], using for example search engine results as proxies [23]. In addition, the process applied in these projects can differ significantly, and several elements of both process and infrastructure could have an impact. For example, [19] has used a sample of projects from SourceForge.net to uncover whether the process maturity has had any on success of open source projects. In this analysis, the notion of success was based on the downloads achieved, and a relationship to version control, mailing lists and testing strategies was found.

In this paper, we apply the method of Data Envelopment Analysis (DEA) to compare open source projects according to their efficiency in transforming inputs into outputs. For any production process, this efficiency and productivity is a key indicator in comparison to other processes. DEA is a non-parametric optimization method for efficiency comparisons without any need for the user to define any relations between different factors or a production function. In addition, DEA can account for economies or diseconomies of scale, and is able to deal with multi-input, multi-output systems in which the factors have different scales.

Efficiency and productivity in software development is most often denoted by the relation of an effort measure to an output measure, using either lines-of-code or, preferably due to independence from programming language, function points [1]. This approach can be problematic even in an environment of commercial software development due to missing components especially of the output, for example also [15] agree that productivity measures need to be based on multiple size measures. In open source development, there are additional problems which point towards DEA as an appropriate method.

In open source projects, normally the effort invested is unknown, and therefore might need to be estimated [2,16,17], and is also more diverse than in commercial projects, as it includes core team member, committers, bug reporters and several other groups with varying intensity of participation. Besides that, also the outputs can be more diverse. In the general case, the inputs of an open source project can encompass a set of metrics, especially concerned with the participants. So, in the most simple case, the number of programmers and other participants can be used. The output of a project can be measured using several software metrics, most easily the number of LOC, files, or others. This range of metrics both for inputs and outputs, and their different scales necessitates application of an appropriate method like DEA.

The main result of applying DEA for a set of projects is an efficiency score for each project. This score can serve different purposes: First, single projects can be compared accordingly, but also groups of projects, for example those following similar process models, located in different application domains or simply of different scale can be compared to determine whether any of these characteristics lead to higher efficiency.

In a prior paper, DEA has been explored in this context, but with a different dataset mostly relying on in-depth CVS analysis [18]. This has demonstrated that DEA can indeed be applied in this context, and has also shown that neither inequality in contributions, nor licensing scheme nor intended audience have a significant impact on efficiency. In this paper, we will employ the results of a DEA to investigate whether the adoption of communication and coordination tools like mailing lists or particular source code control systems have any impact on efficiency.

2 Data Envelopment Analysis

The principle of the border production function was introduced by Farell for measuring the technical efficiency [12] and enhanced by Charnes, Cooper and Rhodes [6] into the first Data Envelopment Analysis model (the CCR model). The object of analysis the DEA considers is very generally termed Decision Making Unit (DMU). This term includes relatively flexibly each unit which is responsible for the transformation of inputs into outputs, for example hospitals, supermarkets, schools, bank branches and others.

The basic principle of DEA can be understood as a generalization of the normal efficiency evaluation by means of the relationship from an output to an input into the general case of a multi-output, multi-input system without any given conversion rates or same units for all factors. In contrast to other approaches, which require the parametric specification of a production function, DEA measures production behavior directly and uses this data for the evaluation of all DMUs. The DEA derives a production function from mean relations between inputs and outputs (whereby it is only assumed that the relation is monotonous and concave), by determining the outside cover of all production relations (see also Figure 1), while for example a regression analysis estimates a straight line through the center of all production relations. The DEA identifies "best practicing" DMUs, which lie on the production border. A DMU is understood as being efficient if none of the outputs can be increased, without either or several of the inputs increasing or other outputs being reduced, as well as vice versa.

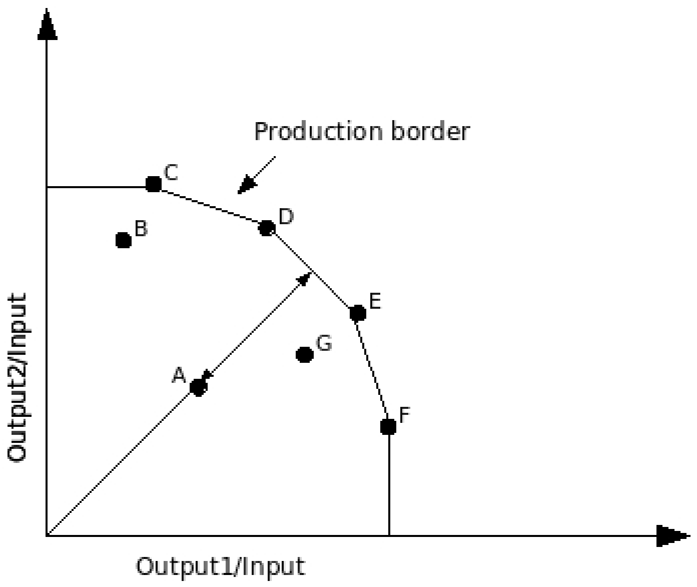


Fig. 1. Data Envelopment Analysis for the case of one input and two outputs with 7 DMUs (A – G), out of which C – F are efficient, and depicting inefficiency of A for which D and E form the reference set

For each DMU an individual weighting procedure is used over all inputs and outputs. These form a weighted positive linear combination, whereby the weights are specified in such a way that they maximize the production relationship of the examined unit, in order to let these become as efficient as possible. The efficiency of an examined unit is limited with 1. That means that no a-priori weightings are made by the user, and that the weights between the DMUs can be different. For each evaluation object the DEA supplies a solution vector of weighting factors and a DEA efficiency score. If this score is equal to 1, then the DMU is DEA efficient. In this context, DEA efficiency means that no weighting vector could be found which would have led to a higher efficiency value. DEA efficient are thus all those DMUs which are not clearly DEA inefficient compared with the others. Any inefficiency can therefore not be ruled out completely. For each inefficient DMU the DEA returns a set of efficient DMUs which exhibit a similar input/output structure and lie on the production border near to the inefficient DMU (reference set, see also Figure 1). Using this information, an idea in which direction an increase in efficiency is possible can be gained.

The first model of the DEA was introduced by Charnes, Cooper and Rhodes [7] and is therefore designated as CCR model. They pose four assumptions for the production possibility set, which are convexity, possibility for inefficient production, constant returns to scale and minimum extrapolation. The different basic models of

the DEA can be divided on the basis of two criteria: This is on the one hand the orientation of the model, on the other hand the underlying assumption regarding the returns to scale of the production process. With input-oriented models the reduction of the input vector maximally possible with the given manufacturing technology is determined, whereas with output-oriented models the maximally possible proportional increase of the output vector is determined. The returns to scale can be assumed either as being constant or variable. With constant returns to scale size-induced productivity differences are considered into the efficiency evaluation, with variable returns to scale the differences are neutralized by the model. The most common example of a model with variable returns to scale is an advancement to the CCR model by Banker, Charnes and Cooper, the BCC model [3]. This model includes an additional measuring variable in the fundamental equation to capture rising, falling or constant returns to scale.
In the area of software development, DEA was so far only rarely applied. Banker and Kemerer use this approach in order to prove the existence of both rising and falling returns to scale [4]. Banker and Slaughter use the DEA in the area of maintenance and enhancement projects [5]. It can be proven that rising returns to scale are present, which would have made a cost reduction of around 36 per cent possible when utilized. An investigation of Enterprise Resource Planning (ERP) projects was done by [20], using 30 SAP R/3-projects of a consulting firm for the application of the DEA. [14] gives an in-depth discussion on the application of DEA in software development.

3 Data Selection and Set

Based on the date January 8th 2007, we selected the thirty most often downloaded projects from SourceForge.net, as presented by the website based on the past 7 days. This statistic is updated daily, the current standings can be seen anytime from the respective web page[1]. This was done in order to arrive at a relatively homogeneous set of projects. Potential problems and pitfalls in using this approach are described in the following.

For each of these projects, a number of variables was retrieved from the respective homepage. We define and use the following variables in this study, with binary variables later on employed for distinguishing between groups of projects:

- Project: This simply gives the project's name.
- Donations: This binary variable shows whether the project has activated the donations feature.
- GNU-style licence: This binary variable codes whether a project is under a GNU GPL licence (true) or not (false), to give an impression of whether a strict copyleft-scheme is followed by the project.

[1]http://sourceforge.net/top/toplist.php?type=downloads_week

- Audience: Again, the intended audience of a project is coded as a binary variable, depending on whether the intended audience is developers or system administrators (true) or not (false).
- Age: The age of the project in years, which is computed based on the year in which the project was registered on SourceForge.net (using 2007 minus registration year).
- Developers: This is the number of developers as reported by the project's page on SourceForge.net.
- Downloads: This is the number of downloads of the project within the last 7 days, as given by the respective statistics page described above.
- Status: The development status from the web page. This is assigned by the project's administration, and has seven possible values, reaching from planning, pre-alpha, alpha, beta to production/stable and mature, and to inactive.
- Translations: The number of different translations available, from the project's page, with all languages counted equally as one.
- Operating Systems: As translations, but with the respective operating systems (or families thereof, e.g. all Windows versions are counted as one).
- Tracker: This binary variable codes whether the project employs the tracker service of SourceForge.net (true in that case).
- Tracker total: This is the total number of entries summed over all different active trackers of a project.
- Mailing list: This binary variable codes whether the project employs the mailing list service of SourceForge.net (true in that case).
- Mailing list total: This is the total number of postings summed over all mailing lists of a project.
- Forum: This binary variable codes whether the project employs the forum service of SourceForge.net (true in that case).
- Forum total: This is the total number of messages summed over all different active forums of a project.
- Tasks: This binary variable codes whether the project employs the task service of SourceForge.net (true in that case).
- Tasks total: This is the total number of tasks (in any status like open or closed) summed over all different subprojects of a project.
- CVS: This binary variable codes whether the project employs a CVS repository (true in that case).
- CVS commits: The total number of commits to the CVS repository as given on the project's page.
- SVN: This binary variable codes whether the project employs an SVN repository (true in that case).
- SVN commits: The total number of commits to the SVN repository as given on the project's page.
- Size: The size in byte of software offered, summing over all packages of the project in the latest release.

The first, and a major source of possible threats is construct validity. Several measures used for conceptualizing different aspects for the following analyses might be problematic and need to be discussed in this context. First is the notion of developer, which is taken directly from the web page. In some projects, people could be contributing code without relevant account, which sometimes is only granted to long-time participants, by sending it to one of those persons who then does the actual commit. Therefore, the number of developers might actually be higher than the number reported here. This fact is very problematic to check. In a case study of the OpenACS project under participation of project insiders and using the strict standards for CVS comments, [11] have found that only 1.6% of revisions pertained to code committed for someone without CVS privilege. Other metrics suffer from similar possible problems, for example a project might have existed before it was registered on SourceForge.net, and also the size might be affected by several factors like different compression algorithms employed. In addition, several parts of the coordination and communication tools might be in use, but not opened to the public, and thus disregarded in this context, or tools completely distinct from the platform might be employed. This is for example true for the source code versioning systems in our dataset. Also [13] give an overview of problems associated with mining data from Sourceforge.net. Lastly, the external validity of the results depends on the selection of an appropriate dataset. In our case, the approach is still mostly exploratory, but using the definition above, a coherent dataset was aimed at. For, example, this shows in the intended audience, which in no case is developers or system administration alone, or the fact that all projects save one use a GPL-licence. Table 1 gives descriptive statistics for some relevant metrics.

Table 1. Descriptive Statistics of Dataset (N=30)

	Median	Mean	Std.Dev.	Min.	Max.
Downloads	97,682.00	285,507.53	538,548.91	63,122	2,457,185
Developers	9.00	10.23	9.53	1	39
Status	5.00	n/a	n/a	4	6
Age	5.00	4.87	1.76	2	8
Translations	1.50	7.90	11.19	1	35
Size	21,220K	106,780K	232,072K	2,832K	998,118K
Tracker total	175.00	1,313.73	4,095.57	0	22,527
Mailing list total	29.00	10,070.13	31,792.62	0	169,574
Forum total	0.00	3,659.40	9,520.53	0	44,272
Tasks total	0.00	2.33	9.83	0	52

4 Analysis and Results

Based on the data set and variables as described above, we set up an DEA with the following parameters, using the program accompanying [8]: The first choices to be taken concern the definition of input and output factors, as well as the model to be applied. Based on the literature on DEA in the context of IT-projects [4,5,14,20], variable returns to scale are selected. Regarding the orientation of the model, an output-orientation might seem more appropriate. Given a certain input which can be acquired, i.e. participants attracted, the output is to be maximized. According to this reasoning, the BCC-O model is applied.

Regarding the definition which factors are to be used as inputs and outputs, it is to be considered that with an increase in the number of factors more DMUs are estimated to be efficient. Also the availability of factors in the data set limits the possibilities. In this case, we selected to use the number of developers and years of existence as inputs, downloads, size, status and translations as outputs. Naturally, this selection is based on the available data, and could be changed.

For an overview of the results, see Table 1. In this table, statistics on the efficiency scores in the total population are given. Overall, 11 different projects have been classified as DEA efficient, the mean efficiency score with 0.922 seems relatively high. For each efficient project, the number of times it appears in the reference sets of non-efficient projects is also given. This can be used as an indicator of the relative importance of this project in determining efficiency scores.

Table 2. Results of Applying DEA to the dataset

No. of DMUs	30
Average	0.922
Std. Dev.	0.096
Median	0.953
Minimum	0.706
Maximum	1.000
Number of DEA-efficient DMUs	11
Frequency in Reference Set	
Peer set	**Frequency**
eMule	8
Ares Galaxy	0
Azureus	3
GTK+ and The GIMP installers for Windows	10
eMule Plus	0
emule Xtreme Mod	15
Portable Apps	6
CDex	16
Gaim	2
MediaCoder	0
WinSCP	0

As one of the results is an efficiency score for each project, we can now use this score for analysing potential effects on this efficiency. As explanatory variables, information concerning the communication and coordination tools employed by the projects is used. As a start, correlations between the efficiency scores and these metrics are explored to uncover any relationships. All of the following analyses were performed using R (version 2.4.0), a free software environment for statistical computing and graphics. Specifically, tracker, mailing lists, forums, tasks and both CVS and SVN were explored as possible influences. In addition, a new metric was computed summing up the binary variables depicting whether or not a tool was employed, to give an indication of the overall diversity of a project in this context. This shows out of a maximum of 6 a mean of 2.53 with median 3 and 1.48 standard deviation.

The results are not conclusive: Regarding correlation coefficients, these are mostly small (below 0.3) and for all tools except forums negative. Also the correlation to the overall number of tools employed is with -0.257 negative. Using non-parametric Mann-Whithney U-tests, these results were tested for statistical significance: The negative relationships with overall tool usage ($p<0.01$), CVS ($p<0.05$), tasks ($p<0.01$), and the positive relationship with forum employment ($p<0.01$) are statistically proven. The results from [18] regarding licensing scheme and audience could not be checked due to minimal respectively no variance in these attributes, the inequality in contributions was not available in this data set.
These results seem rather surprising, given that [19] found a relationship between process maturity and success, but there are two different explanations: First, the tools as provided by SourceForge.net are not giving relevant help to the projects employing them, so projects using other tools, or even none at all for a given task perform better. The second explanation would be that all the tools for communicating with users and potential co-developers are more of a hindrance to efficient software development, detracting attention and time from the developers, which might be better spent on actual development work. Naturally, the view on this might also depend on the output factors included: Employing mechanisms like bug tracking might help to achieve higher quality in the released software, and it is unclear whether this effect is currently incorporated. Naturally, it could be assumed to higher quality projects achieve a higher number of downloads, but including quality aspects in the list of output factors might give additional insights.

5 Conclusion and Future Research

In this paper, we have used a method to compare the efficiency of open source projects to analyse potential impacts of different communication and coordination tools. The method used is the DEA, which is well-established in other fields and offers several advantages in this context as well: It is a non-parametric optimization method without any need for the user to define any relations between different factors or a production function, can account for economies or diseconomies of scale, and is able to deal with multi-input, multi-output systems in which the factors have different scales. Using a data set of 30 open source project retrieved from SourceForge.net, we have demonstrated the application of DEA. Results show that the efficiency of the projects is in general relatively high with low variance. Regarding the effects of tool employment on the efficiency of projects, the results were surprising: Most of the possible tools, and overall usage, showed a negative relationship to efficiency. This could be either due to more efficient tools being available elsewhere, or a negative influence of all activities except software development per se.

In future research, additional work has to be done on arriving at a common understanding of input and output factors and their definitions. For example, using the size in bytes instead of lines-of-code might be problematic, but on the other hand captures other output aspects like audio or others as well. Also the selection of projects to be included might be worked on, to preclude projects without real development work, which only serve as assemblers of others. Further, additional analyses based on the results using other project characteristics would be of high interest. For example, the definition of different process models would be of high interest for efficiency comparisons. These could also include comparisons within application areas, different project scales, and comparisons to commercial or mixed-mode development projects.

References

[1] Albrecht, A.J., & Gaffney, J.E. (1983). Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering*, 9(6), 639-648.

[2] Amor, J.J., Robles, G., & Gonzalez-Barahona, J.M. (2006). Effort Estimation by Characterizing Developer Activity. In *Proceedings 8th International Workshop on Economics-Driven Software Engineering Research (ICSE 2006)*, Shanghai, China.

[3] Banker, R.D., Charnes, A., & Cooper, W. (1984). Some Models for Estimating Technical and Scale Inefficiencies in Data Envelopment Analysis. *Management Science*, 30, 1078-1092.

[4] Banker, R.D., & Kemerer, C. (1989). Scale Economies in New Software Development. *IEEE Transactions on Software Engineering*, 15(10), 416-429.

[5] Banker, R.D., & Slaughter, S.A. (1997). A Field Study of Scale Economies in Software Maintenance. *Management Science*, 43(12), 1709-1725.

[6] Charnes, A., Cooper, W., & Rhodes, E. (1978a). *A Data Envelopment Analysis Approach to Evaluation of the Program Follow Through Experiments in U.S. Public School Education* (Management Science Research Report No. 432). Carnegie-Mellon University, Pittsburgh, PA.

[7] Charnes, A., Cooper, W., & Rhodes, E. (1978b). Measuring the Efficiency of Decision Making Units. *European Journal of Operational Research*, 2, 429-444.

[8] Cooper, W., Seiford, L., & Tone, K. (2000). *Data Envelopment Analysis: A Comprehensive Text with Models, Applications, References and DEA-Solver Software*, Boston, MA: Kluwer Academic Publishers.

[9] Crowston, K., Annabi, H., & Howison, J. (2003). Defining Open Source Software Project Success. In *Proceedings of ICIS 2003*, Seattle, WA.

[10] Crowston, K., Annabi, H., Howison, J., & Masango, C. (2004). Towards A Portfolio of FLOSS Project Success Measures. In *Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering (ICSE 2004)*, Edinburgh, Scotland.

[11] Demetriou, N., Koch, S. & Neumann, G. (2006). The Development of the OpenACS Community. In Lytras, M. & Naeve, A. (eds.) *Open Source for Knowledge and Learning Management: Strategies Beyond Tools*, Hershey, PA: Idea Group.

[12] Farell, M.J. (1957). The Measurement of Productive Efficiency. *Journal of the Royal Statistical Society*, Series A 120(3), 250-290.

[13] Howison, J. & Crowston, K. (2004). The perils and pitfalls of mining SourceForge. In *Proceedings of the International Workshop on Mining Software Repositories*, pp. 7-11, Edingburgh, Scotland, UK.

[14] Kitchenham, B. (2002). The question of scale economies in software - why cannot researchers agree? *Information & Software Technology*, 44(1), 13-24.

[15] Kitchenham, B., & Mendes, E. (2004). Software Productivity Measurement Using Multiple Size Measures. *IEEE Transactions on Software Engineering*, 30(12), 1023-1035.

[16] Koch, S. (2004). Profiling an open source project ecology and its programmers. *Electronic Markets*, 14(2), 77-88.

[17] Koch, S. (2005). *Effort Modeling and Programmer Participation in Open Source Software Projects* (Arbeitspapiere zum Tätigkeitsfeld Informationsverarbeitung, Informationswirtschaft und Prozessmanagement, Nr. 03/2005). Wirtschaftsuniversität Wien, Vienna, Austria.

[18] Koch, S. (to appear). Measuring the Efficiency of Free and Open Source Software Projects Using Data Envelopment Analysis. In Sowe, S.K., Stamelos, I. and Samoladas, I. (eds.): *Emerging Free and Open Source Software Practices*.

[19] Michlmayr, M. (2005). Software Process Maturity and the Success of Free Software Projects. In Zielinski, K. and Szmuc, T. (eds.): *Software Engineering: Evolution and Emerging Technologies*, pp. 3-14, IOS Press, Amsterdam, The Netherlands.

[20] Myrtveit, I., & Stensrud, E. (1999). Benchmarking COTS Projects Using Data Envelopment Analysis. In *Proceedings of 6th International Software-Metrics-Symposium,* pp. 269-278, Boca-Raton.

[21] Stewart, K.J. (2004). OSS Project Success: From Internal Dynamics to External Impact. In *Collaboration, Conflict and Control: The 4th*

Workshop on Open Source Software Engineering (ICSE 2004), Edinburgh, Scotland.

[22] Stewart, K.J., & Ammeter, T.A. (2002). An Exploratory Study of Factors Affecting the Popularity and Vitality of Open Source Projects. In *Proceedings of ICIS 2002*, Barcelona, Spain.

[23] Weiss, D. (2005). Measuring Success of Open Source Projects Using Web Search Engines. In *Proceedings of the 1st International Conference on Open Source Systems*, pp. 93-99, Genoa, Italy.

Innovation in Open Source Software Development: A Tale of Two Features

John Noll

Santa Clara University
Computer Engineering Department
500 El Camino Real, Santa Clara, CA USA
jnoll@cse.scu.edu

Abstract. Open Source Software Development appears to depart radically from conventional notions of software engineering. In particular, requirements for Open Source projects seem to be asserted rather than elicited.
This paper examines two features of selected open source products: "tabbed browsing" as realized in the Firefox web browser, and "edge magnetism" found in the Gnome desktop environment's Metacity window manager. Using archives of mailing lists and issue tracking databases, these features were traced from first mention to release, in attempt to discover the process by which requirements are proposed, adopted, and implemented in their respective Open Source projects. The results confirm the importance of user participation in Open Source projects.

1 Introduction

A common view states that open source projects begin as the need for "scratching a developer's personal itch [20]," in other words, to fill a need that is not addressed by a current commercial or free product. This lead developer then becomes the shepherd of a growing community of volunteers who contribute programming labor to the project until it evolves into a useful product. But once started, how do open source projects continue to innovate? How do they stay competitive with, and even dominate, their commercial competition? Empirical studies of open source software development suggest that open source projects follow different processes than traditional textbook approaches [24].

This paper examines the history of two features of two open source software products — "tabbed browsing" in the Mozilla project's Firefox web browser, and the "edge magnetism" behavior of the Gnome desktop environment's Metacity window manager — to see how new features are proposed, debated, and finally adopted for release. This examination shows that innovation occurs in a variety of ways, sometimes following a conventional software engineering approach, other times resembling Raymond's itch scratching. Perhaps most interestingly, these differing processes can coexist in a single project, providing multiple sources for innovation.

Krishnamurthy's study of projects hosted by *Sourceforge.net* supports Raymond's idea of project initiation: of the 100 most active projects marked *mature* in the Sourceforge coding scheme, the median number of developers on a project was four [12],

Please use the following format when citing this chapter:

Noll, J., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 109–120.

confirming that many projects start small, but also suggesting that many projects never grow beyond a handful of developers.

But the itch scratching explanation does not fit all open source projects, nor does it explain how at least some open source software grows and evolves past the initial release into feature-rich products; many open source projects continue to evolve into comprehensive products that have capabilities far beyond their original conception. They have excellent quality and sometimes dominate their markets.

For example, the Apache web server has grown to include numerous innovative features including a built-in Java virtual machine, Perl interpreter, and database access, that extend Apache's functionality far beyond its original purpose as an HTTP server of HTML home pages. The Apache server is reported to host the majority of the world's web sites [29, 15]. The Apache foundation now includes projects far beyond the Apache web server product, encompassing such diverse elements as XML processing libraries and parsers, software build tools, and email processing software [5].

Open source projects based on formerly proprietary products also continue to innovate. The Firefox web browser is considered to be one to the most secure web browsers available and is actually gaining market share compared to its chief commercial competitor (Internet Explorer) [10, 22]. Firefox evolved from the Mozilla codebase, which in turn is a descendant of the Netscape Navigator code that was released as open source by Netscape Communications, Inc. [6]

Similarly, OpenOffice was created as an open source version of Sun Microsystem's StarOffice commercial office automation product [18]; it now has many innovative features including an XML-based storage format and plug-in "channels" for importing other file formats. In both cases, these products have evolved into feature-rich products that take them far beyond mere copies of their commercial competitors.

The goal of this paper is to discover how mature Open Source Software projects develop new features. Toward this end, the history of Firefox's tabbed browsing and Metacity's edge magnetism was traced by examining discussions of each feature in project mailing lists, web logs, issue tracking systems, and other on-line forums. The results provide a snapshot of how new ideas are incorporated into products, providing further insight into Open Source development processes.

The next section describes the projects and features studied. Following that is a discussion of observations and their implications. The last sections present related work and conclusions.

2 Background

Two open source software products were chosen to study: the Firefox web browser, and the Metacity window manager for the Gnome desktop environment. These projects represent different types of open source projects: Firefox has roots in a proprietary product (Netscape Navigator), while the Gnome project was open source from its inception. Firefox has at least one serious proprietary competitor, while Gnome is targetted for Unix and Linux platforms, and thus its chief competitor is another open source product (KDE). The Firefox architecture incorporates extension mechanisms that allow programmers to add new functionality without modifying the core source code. Gnome, being an environment for managing multiple applications on a user's desktop, provides services and libraries that programmers use to create applets and applications that may cooperate with other Gnome programs. Metacity is one such program, created with simplicity as a goal and thus providing minimal customization options.

These two projects also have significantly different organization and management structure. The Mozilla development organization has a substantial co-located workforce that can hold traditional face-to-face meetings; these are used for release planning. Gnome is a "pure" open source project with a governing board and foundation to accept contributions from commercial firms. The project management and labor remain completely distributed. Open Office falls in between these extremes: it receives significant support from Sun microsystems, including funds and labor, but the project management and programming effort are widely distributed.

In order to understand how each project develops new functionality, a single feature of each product was chosen from a current release:

1. "Tabbed browsing" in the Firefox web browser, which allows users to open and manage multiple web pages within a single browser window.
2. "Window magnetism" (also called "edge resistance" and "window snapping") in Gnome's Metacity window manager, which helps users place windows on the desktop in a "tiled" arrangement.

Archives of project discussion forums were then examined to determine when the feature was first proposed, how it was debated, and when it was ultimately adopted to be delivered in a specific release. Because a typical open source project involves widely distributed programmers, testers, and users, management and technical discussions are conducted using digital communication technology such as email lists, news groups, issue tracking systems, and increasingly "chat" channels and web logs. With the exception of chat channels, these discussions are archived and made available to the general public, as a way of preserving the history of design decisions and to provide a means for newcomers to understand how a given project conducts its business. Discussions conducted via chat channels are sometimes archived as well, but this practice does not seem to be as common as archiving other media.

The details of each feature are explained in the following sections.

2.1 Firefox Tabbed Browsing

Firefox is a web browser developed by the Mozilla foundation. The Mozilla foundation was created to oversee the continued evolution of Netscape Navigator and Communicator when Netscape Communications, Inc. decided to transition development of their web browser and related software to an open source model. The Mozilla foundation's orginal product, *Mozilla*, is an integrated web browser, email client, and web page composer; while it is still being distributed and maintained, the Mozilla foundation's long term strategy is to replace the monolithic Mozilla product with separate, single-purpose programs: the Firefox web browser, and the Thunderbird email client [1].

Firefox has been highly successful, earning praise for its innovative features as well as robustness [10, 22].

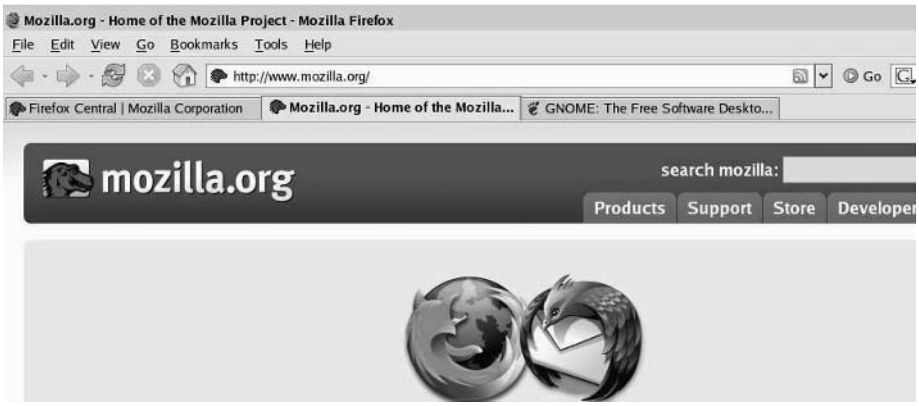

Mozilla.org - Home of the Mozilla Project - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://www.mozilla.org/
Go
Firefox Central | Mozilla Corporation
Mozilla.org - Home of the Mozilla...
GNOME: The Free Software Deskto...
mozilla.org
search mozilla:
Products
Support
Store
Developer

Fig. 1: Tabbed Browsing in Firefox

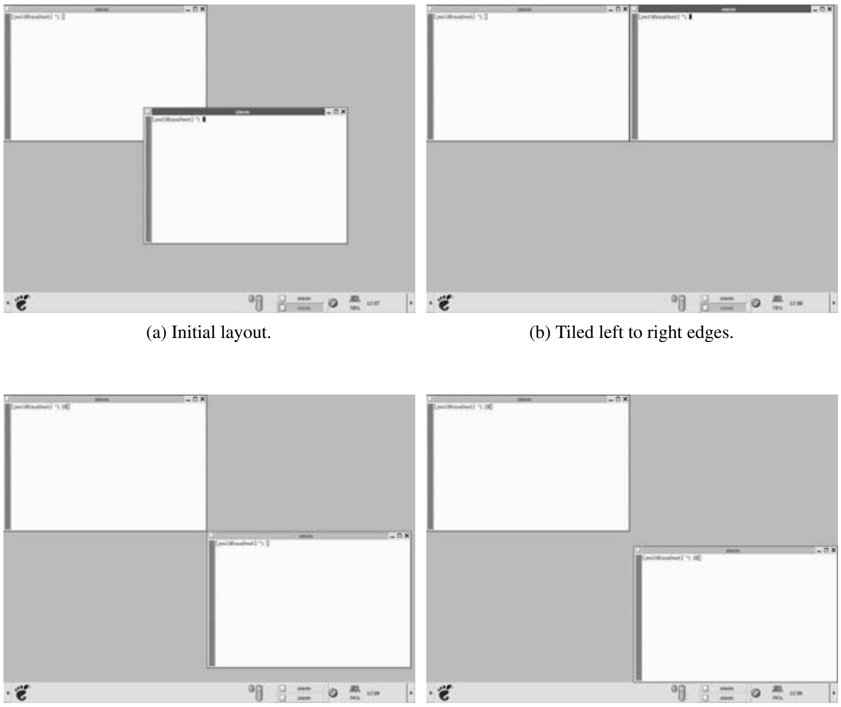
Tabbed browsing is a feature available in several comtemporary web browsers that allows the presentation of multiple web pages in a single window (Figure 1). Each page is identified by a "tab" resembling the label tab of a paper file folder; users can switch the window's display from one page to another by clicking on the page's associated tab. In the current version of Firefox (version 1.5 as of this writing), the pages can be re-ordered by dragging the tabs to the right or left; new pages can be opened in an existing tab, a new tab, or a completely new browser window.

2.2 Metacity Edge Magnetism

Metacity is the default window manager for the Gnome desktop environment. Unlike Microsoft windows and other window-based user interfaces that are part of the operating system, the X Window System — the windowing platform on which Gnome runs — is a set of user-space programs that work together to create the windowing environment. A *window manager* for the X window system is the program that is responsible for creating windows and decorating them with borders and buttons to minimize, maximize, and close windows; the window manager is also responsible for moving windows in response to mouse or keyboard events, and to provide keyboard shortcuts. As such, the window manager has a significant effect on the appearance and operation of a user's desktop.

Because a window manager is separate from the operating system, users are free to choose any window manager that suits their needs and taste; a variety of window managers for the X window system have been created to satisfy different user requirements.

Metacity was created as a replacement for Sawfish, the previous default window manager for the Gnome desktop environment. Metacity could be seen as a reaction to Sawfish's complexity and lack of stability; Sawfish was highly configurable, having a built-in Scheme interpreter, but had a high fault rate which many considered to be a side effect of its rich functionality. This excerpt from a posting to the Gnome support forum illustrates [3]: "... people praise Sawfish features yet they hate the massive amount of bugs. These two things go hand in hand. There is a reason Sawfish is practically not maintained anymore." Metacity was designed to include the minimum set of useful features, with minimal configurability; the emphasis was on robustness rather than richness.



(a) Initial layout. (b) Tiled left to right edges.

(c) Tiled corners. (d) "Snapping" to Gnome toolbar.

Fig. 2: Window edge magnetism in Metacity.

Window *edge magnetism*, also called *edge resistance* or *snapping*, makes positioning a window on the desktop easier by changing the behavior of a window when it is moved near another window: the moving window will resist overlap of another window, and will try to align its edges with that window (Figure 2). The windows behave as if they were magnetized; like real magnets, they resist certain arrangements and "snap" to a tiled arrangement where window sides (Figure 2b) and corners (Figure 2c) are aligned, and windows do not overlap other windows. This makes organizing the desktop for maximum window visibility easier.

3 Observations

3.1 Tabbed Browsing

The first reference to tabbed browsing on a Mozilla-oriented newsgroup appears to be June 23, 1999, when a Mozilla user posted a note to `netscape.public.mozilla.wishlist` requesting the ability to download a new web page while viewing another page [16]:

> One thing that I would really want to see is the ability to open a link in the new window in background (i.e. the focus should remain in my current window, and new window should load silently, without bothering me until it is ready and I am ready to read it).

This spawned a discussion of the merits of tabbed browsing; the following poster is referring to the Opera web browser's tabbed browsing feature: [7]

> Have you tried tabbed browsing? Now that I've tried it, I won't go back to windows everywhere. The idea is that pages have their own tabbed windows. Instead of juggling windows, you just click their tabs. The beauty part is new pages open in the background, just as you requested. The tab tells you when the page is done loading. Then you just click over. Shweet!

H.J. van Rantwijk claims to have proposed addition of tabbed browsing to Mozilla on the Mozilla developer's chat forum (`#mozilla` at `irc.mozilla.org`), but got no positive response. The Mozilla foundation does not make public its archives of chat channels, so this claim is difficult to verify. Regardless, using Mozilla's extension mechanism, he was able to implement and distribute this functionality anyway. The result, an extension called MultiZilla, implemented the first tabbed browsing functionality for the Mozilla browser. This project began in April 2000 [28].

Subsequently, David Hyatt implemented tabbed browsing for Mozilla (version 0.97, released in December, 2001) directly, influenced by MultiZilla. But this implementation was done from scratch without using any code from MultiZilla [27]. Hyatt went on to create the Firefox browser (with Blake Ross); the first implementation of Firefox (then called "Phoenix") to feature tabbed browsing was the 0.3 milestone of October, 2002, which led to the first official Firefox product release (1.0) in November of 2004 [9].

What this suggests is that new features can follow several paths from suggestion to release. Tabbed browsing first appeared in Mozilla as an extension written by a volunteer who was unsuccessful getting acceptance from the core Mozilla developer community. The extension proved to be useful enough that one of the core developers incorporated it into the main Mozilla code base, which eventually led to its inclusion in Firefox.

An important aspect of this path is that it is enabled by the Firefox architecture: because Firefox's user interface implementation allows user interface behavior to be defined using a specification language called XUL [2], writers of extensions to the user interface don't have to change any of the core code; rather, they write a new XUL specification.

3.2 Edge Resistance

Edge resistance was available in other window managers, including the existing Gnome window manager (Sawfish), when Metacity was released. Many users missed this feature in Metacity; the following quote[1] from a posting to the `gnomesupport.org` forum illustrates [13]

> Recently, I updated my box to Redhat 9.0 and dropped Sawfish in favour of Metacity. However, there are two things I used to use in Sawfish I am not able to use in Metacity:
> - Configure keys to move the cursor.
> - Switch on windows "magnetism" to help a easy windows placement.
> I didn't find any option, does anybody here know where to touch?

This comment sparked a debate on the GNOME support forum (gnomesupport.org) centered on the tension between feature richness and maintainability. Another poster echoed the above sentiments [30]:

> Fly has a point about the usability of Metacity. I understand the complaints about the "bloat" in Sawfish, but as far as memory footprint is concerned, there is very little difference between Sawfish and Metacity. To claim that including sensible features is adding bloat is just feeding us a line of bullshit. I've been using Metacity since Gnome 2.0, mainly because it is now inconvenient to manage themes properly with Sawfish in the picture, but it would be very nice if Metacity would remember window sizes and placement. As far as I'm concerned, that is a window manager's job, and Metacity is clearly shirking that job.
> If it wasn't for those apps that remember their own window geometry, I would be getting quite fed up with Metacity by now.

Then, "Fly" followed his earlier posting with some general comments

> ... I understand that many features in Sawfish [are] excessive or unrelated to WM, but 80% of Sawfish features very useful and I need it - you not?

In response, "Dbrody" (a 'guru' on this forum) said

> But if only $> 5\%$ of people need $> 80\%$ of those features then you have just proved that it is bloat. Bloat dosn't mean memory foot print. That is NOT what anybody cares about the extra 1k of ram. Bloat means the maintainer needs to start maintaining more features. More bug reports. More tweaking of those

[1] Original spelling and proper name capitalization in quoted excerpts has been corrected.

> advanced features. etc... etc... Metacity is not even 1.0 yet. There are many changes that are planned to go into Metacity but haven't yet because things go a little slowly, or because it will make Metacity incompatible with themes and so forth.
> Also, many of these feature can be done with external programs, like devilspy. Certainly things like edge flipping, advanced focus management, etc... are easily done using libwnck and a little hacking.

This debate is interesting because it takes place in a public forum where anyone - users, developers, interested third parties - can participate. The discussion of requirements is therefore completely transparent, and also recorded in significant detail, so that the rationale behind any decision can be discovered later if necessary.

The creator of Metacity agreed on the usefulness of edge magnetism almost a year earlier; he filed the following Request for Enhancement (RFE) in the Gnome project's issue tracking system in May 2002 [19]:

> Add some kind of mild "attraction" to window/screen edges, perhaps only after a timeout. Need to experiment.

This entry stimulated a lengthy discussion of exactly how this behavior should work, which continued for over three years until the feature was incorporated into the release code-base in November of 2005.

Again, the discussion takes place in a public forum (the Gnome issue tracking database is readable by anyone, and anyone who registers can post issue reports or comments), and is recorded for future reference.

The history of edge magnetism in Metacity represents a combination of two phenomena that appear to be characteristics of Open Source development projects. First, the significant, lively participation by users of Metacity in the discussion about the merits of and desire for edge magnetism are an example of the essential role that users play in Open Source projects [4]. Second, Havoc Pennington's RFE is an example of an asserted requirement: the developer of a product has stated the need for a particular feature; this is consistent with observations made by other researchers [8, 23].

3.3 Discussion

Gnome and Metacity closely resemble the common notion of open source development, where features are proposed in an on-line forum (newsgroup, mailing list, issue database), debated by users and programmers, and ultimately adopted or rejected. A feature may be adopted by virtue of having a working implementation, regardless of its merits.

In contrast, Firefox follows an almost traditional process involving regular face-to-face release planning meetings. But Firefox's extension mechanism allows features that are initially rejected to "prove" their worth by demonstrating adoption by real users.

The openness of various communication channels employed by open source projects enables and encourages enthusiastic participation by users of the product, as well as developers. This provides early feedback about a product's functionality and shortcomings, as well as a way to capture users' ideas and needs.

Likewise, open issue tracking mechanisms provide a way for end users to voice their concerns about product functionality, and participate in the discussion about resolutions and enhancements. This has advantages for both the developers and users: the developers can potentially seek clarification through the discussion feature of issue tracking systems like Bugzilla, while users seem to develop a sense of ownership as they see their concerns actively considered and their participation encouraged.

4 Related Work

Studies of open source software projects address a wide range of topics from economics [29] to maintainability [25].

A number of case studies have examined open source development processes, including those employed by Apache and Mozilla [14, 21, 24]. In particular, Reis and de Mattos Fortes, in their study of Mozilla development processes, report that high level requirements are specified by the `mozilla.org` management, but all development on the Mozilla code base originates with a "bug" report, which might be submitted by another developer, tester, or end user [21]. These reports may document some product failure, or a request for enhancement.

Feller and Fitzgerald note that users are a "critical feature" [4, 10] of OSSD projects, as the source of new requirements. Scacchi has made several studies of requirements acquisition in open source software development; he observes that requirements "emerge" from on-line discussions which are usually open forums, rather than through traditional requirements elicitation processes, but that this emergent process, though less formal, is also effective [24, 23]. He also notes that requirements are "asserted" after the fact; other researchers have echoed this observation. In particular, German reports a similar situation in the Gnome project [8]. This seems to contradict conventional understanding that cites failure to understand requirements as a major source of software project failure.

But Trudelle notes, in his discussion of lessons learned from experience working on Mozilla, that this approach led to rework of some of the Mozilla implementation in response to user-submitted bug reports; his view is that this rework could have been avoided with traditional up-front requirements analysis and design activities [26]. Henderson echoes this view, claiming that open source projects do not employ "requirements elicitation," but that this could (and should) be easily added to open source processes [11]. Further, Nichols and Twidale observe that usability requirements are not captured well by OSSD projects, due to the mismatch between developers and users; their view is that the OSSD approach of "coding as early as possible" violates "good interface design." [17]

These observations run counter to the prevailing OSSD view that de-emphasizes formal design and requirements gathering. Trudelle's view — that OSSD projects need an overarching UI design and design function — seems to contradict the current success of Firefox, which is widely recognized as among the most innovative web browsers. In particular, Nichols and Twidale's assertion that "commercial software establishes the state of the art" [17] seems to be contradicted by Opera and Firefox, both of which included UI features (tabbed browsing, for example) well before Internet Explorer.

5 Conclusions

Much has been made of the advantages open source development might give to commercial for-profit enterprises, including high quality, free labor, and quick response to critical failures. But the observations presented above reveal some practices that could be useful to any software development effort, including traditional closed source products:

1. Open communication channels between users and developers. This gives users a greater stake in the future of the product, and provides feedback without the overhead of conducting surveys or convening focus groups.
2. Extension mechanisms that allow users with programming skills to demonstrate ideas by contributing working functionality.
3. Alternate paths for ideas to become released features.

Open source projects are far from uniform in their process for conceiving and realizing new features. But they seem to share a common aspect — close involvement of end users in the development process — that is less common in conventional development environments.

Acknowledgments

This work was supported in part by a grant from the School of Engineering at Santa Clara University. No endorsement is implied.

References

1. Alex Bishop. Major roadmap update centers around Phoenix, Thunderbird; 1.4 branch to replace 1.0; changes planned for module ownership model. *MozillaZine (online)*, April 2 2003. `http://www.mozillazine.org/articles/article3042.html`.
2. Peter Bojanic. The joy of XUL. Web page, cited september 6, 2006., Mozilla Foundation, June 2006. `http://developer.mozilla.org/en/docs/The_Joy_of_XUL`.
3. Dbrody. no title. `http://gnomesupport.org/forums/viewtopic.php?t=3603&highlight=&sid=c5f4%e5ae34765db22bac227d7f8b17cb`, September 2003. Posting to the Gnome desktop user support forum.
4. Joseph Feller and Brian Fitzgerald. A framework analysis of the open source software development paradigm. pages 58–69, 2000.
5. The Apache Software Foundation. About the Apache HTTP server project. `http://httpd.apache.org/ABOUT_APACHE.html`. Web page, cited January 16, 2007.

6. The Mozilla Foundation. About the Mozilla foundation. http://www.mozilla.org/foundation/, November 2006. Web page cited January 16, 2007.
7. Gboone. Open new window in background (tabbed browsing). http://groups.google.com/group/netscape.public.mozilla.wishlist/tree/br%owse_frm/thread/ef62c3307e2a7a32/4ec071eae14082ff?rnum=1&hl=en&_done=%2Fgroup%%2Fnetscape.public.mozilla.wishlist%2Fbrowse_frm%2Fthread%2Fef62c3307e2a7a32%2F%4ec071eae14082ff%3Ftvc%3D1%26hl%3Den%26#doc_4b33ef52c30564cf.
8. Daniel M. German. GNOME, a case of open source global software development. In *Proceedings of the 6th International Workshop on Global Software Development*, Portland, OR USA, May 2003.
9. Ben Goodger. Firefox 1.0 roadmap. http://www.mozilla.org/projects/firefox/roadmap-1.0.html, 2004. Web page describing release history of Firefox, cited March 1, 2007.
10. Steve Hamm. A Firefox in IE's henhouse. *Business Week*, September 17 2004.
11. Lisa G. R. Henderson. Requirements elicitation in open-source programs. *CrossTalk - The Journal of Defense Software Engineering*, 13(7):28–30, July 2000. http://www.stsc.hill.af.mil/crosstalk/2000/07/henderson.html.
12. Sandeep Krishnamurthy. Cave or community?: An empirical examination of 100 mature open source projects. *First Monday*, 7(6), June 2002.
13. Lou. Metacity configuration. http://gnomesupport.org/forums/viewtopic.php?t=3603&highlight=&sid=c5f4%e5ae34765db22bac227d7f8b17cb, August 4 2003. Posting to the Gnome desktop user support forum.
14. Audris Mockus, Roy T. Fielding, and James Herbsleb. A case study of open source software development: The Apache server. In *Proceedings of the 22nd International Conference on Software Engineering*, pages 263–272, Limerick, Ireland, May 2000.
15. Netcraft, Ltd. September 2006 web server survey. http://news.netcraft.com/archives/2006/09/05/september_2006_web_server_%survey.html, September 2006.
16. Vladimir Neyman. Open new window in background. http://groups.google.com/group/netscape.public.mozilla.wishlist/tree/br%owse_frm/thread/ef62c3307e2a7a32/4ec071eae14082ff?rnum=1&hl=en&_done=%2Fgroup%%2Fnetscape.public.mozilla.wishlist%2Fbrowse_frm%2Fthread%2Fef62c3307e2a7a32%2F%4ec071eae14082ff%3Ftvc%3D1%26hl%3Den%26#doc_4ec071eae14082ff, June 23 1999. Message posted to netscape.public.mozilla.wishlist mailing list.
17. David M. Nichols and Michael B. Twidale. The usability of open source software. *First Monday*, 8(1), January 2003.
18. OpenOffice.org. About us: OpenOffice.org. http://about.openoffice.org/index.html, January 2007. Web page, cited January 19, 2007.
19. Havoc Pennington. Bug 81704 - Edge magnetism/resistance/snapping/etc. http://bugzilla.gnome.org/show_bug.cgi?id=81704, May 2002. Request for enhancement (RFE) entered into the Gnome project's issue tracking system.
20. Eric S. Raymond. The cathedral and the bazaar. In *The Cathedral and the Bazaar*. O'Reilly and Associates, October 1999.
21. Christian Robottom Reis and Renata Pontin de Mattos Fortes. An overview of the software engineering process in the Mozilla project. In *Proceedings of the Open Source Software Development Workshop*, Newcastle upon Tyne, UK, February 2002.
22. Rachel Rosmarin. Mozilla Firefox gaining ground on Microsoft IE. *Forbes.com*, August 1 2006.

23. Walt Scacchi. Understanding the requirements for developing open source software systems. *IEE Proceedings – Software*, 149(1):24–39, February 2002.
24. Walt Scacchi. Free and open source development practices in the game community. *IEEE Software*, pages 59–66, January 2004.
25. Stephen R. Schach, Bo Jin, David R. Wright, Gillian Z. Heller, and A. Jefferson Offut. Maintainability of the Linux kernel. *IEE Proceedings – Software*, 149(1), February 2002.
26. Peter Trudelle. Shall we dance? Ten lessons learned from Netscape's flirtation with open source UI development. Technical report, Mozilla.org, 2002. Presented at the Open Source Meets Usability Workshop, Conference on Human Factors in Computer Systems (CHI 2002), Minneapolis, MN. Accessed December 28, 2006.
27. unknown. A guide to Mozilla 1.0. `http://www.mozilla.org/start/1.0/guide/`, 2002. Web page describing release 1.0 of Mozilla.
28. H.J. van Rantwijk. MultiZilla's home page. `http://multizilla.mozdev.org`, February 24 2006. Home page for the MultiZilla project, cited September 6, 2006.
29. David A. Wheeler. Why open source software / free software (OSS/FS, FLOSS, or FOSS)? Look at the numbers! Technical report, dwheeler.com, 2005.
30. WonkoTheSane. Untitled. `http://gnomesupport.org/forums/viewtopic.php?t=3603&highlight=&sid=c5f4%e5ae34765db22bac227d7f8b17cb`, September 22 2003. Posting to the Gnome desktop user support forum.

Corporate Involvement of Libre Software: Study of Presence in Debian Code over Time*

Gregorio Robles, Santiago Dueñas, Jesus M. Gonzalez-Barahona

GSyC/LibreSoft, Universidad Rey Juan Carlos (Madrid, Spain)
{grex,sduenas,jgb}@gsyc.escet.urjc.es

Abstract. Although much of the research on the libre (free, open source) phenomenon has been focused on the involvement of volunteers, the role of companies is also important in many projects. In fact, during the last years, the involvement of companies in the libre software world seems to be raising. In this paper we present an study that shows, quantitatively, how important this involvement is in the production of the largest collection of code available for Linux: the Debian GNU/Linux distribution. By studying copyright attributions in source code, we have identified those companies with more attributed code, and the trend of corporate presence in Debian from 1998 to 2004.
Keywords: open source, libre software, involvement of companies, empirical study, software business

1 Introduction

For companies producing computer programs, libre software[2] is not yet another competitor playing with the same rules. The production of libre software differs from *traditional* software development in many fundamental aspects, ranging from ethical and psychological motivation to new economic and marketing premises, to new practices and procedures in the development process itself.

One of the key differences is the different role of users. While in the *classical* software development environment the development team can be clearly distinguished from the users, most of the libre software projects develop around themselves a community [7]. This community is usually formed by people with many different involvements, from pure users to core developers, including many mixed roles, such as that of users contributing with patches (small modifications) to the code. Therefore, in most libre software projects we may observe a *continuum* of commitment to the project which includes a wide range of occasional contributors.

* This work has been funded in part by the European Commission, under the FLOSSMETRICS (FP6-IST-5-033547) and FLOSSWORLD (FP6-IST-015722) projects.

[2] Through this paper the term "libre software" will be used to refer to code that conforms either to the definition of "free software" (according to the Free Software Foundation) or of "open source software" (according to the Open Source Initiative).

Some software companies have realized this fact, sponsoring and promoting projects with the aim of benefitting from the development of a strong community around them. Some of the most known cases in this realm are Sun Microsystems (OpenOffice.org, OpenSolaris, GNOME, among others), IBM (Apache, Eclipse, etc.) or Apple (Darwin, the kernel of Mac OS X). Be it for this reason or for any other, the involvement of companies in libre software development is strong, and increasing with time.

While there has been some empirical research on self-organized software development in libre software (among others, see [6, 19]) and especially on activities performed by volunteers [12, 17], including their integration process [20], the involvement of software companies in the phenomenon has been rarely attended and if it has been mainly from the point of view of business models, business case studies, and the motivations behind companies [4, 5, 2, 3]. Hence, the focus of these papers can be understood from the perspective of companies wanting to understand, invest or to guide them to successfully get integrated into the libre software phenomenon.

Many of the companies that work with libre software just take already written code and adapt it without providing feedback to the community, but some others actively participate in libre software projects. In general, what these companies are looking for in libre software is to obtain a surrounding user community which serves both as a basic and fast feedback mechanism, but also as a marketing strategy, with the aim of getting software of better quality by letting external brainware access the project's source code, to lower the cost by letting volunteers enhance or fix the software, among others.

The target of this paper is to measure the involvement of libre software companies in libre software, specifically of those that deliver the code to the community. For this, we will analyze the source code available in the Debian GNU/Linux stable distributions, which contain in its latest version more than 10,000 source code packages (usually applications, but also libraries and other components). As the sources of several Debian stable releases are available, we will apply our methodology to five of them, spawning from 1998 to 2005, therefore tracking the evolution of the participation of companies during a period of 7 years.

The rest of this paper is structured as follows: next, the methodology for our empirical study will be presented. In this section, we will present the data sources and the procedures we have used to extract and analyze them. We will discuss why the use of copyright statements is significant for our approach and will include some refinements in our methodology to avoid double counting files. In the following section, we will present the results of applying the methodology to Debian, the largest libre software GNU/Linux distribution. We will provide results over time to compare the evolution of the involvement. Finally, conclusions, limitations and future research opportunities are presented.

2 Methodology

Contrary to popular belief, libre software authors rely on copyright law to enforce their licenses, and therefore include copyright marks in their programs. This is especially true for companies interested in maintaining ownership on the code their (hired) developers have produced. Even when the software is licensed under libre software licenses, the copyright owner has some privileges (such as relicensing under other licenses) which are usually appealing to companies. Companies also tend to have strict policies about copyright notices in source code files.

But it is not only in the interest of companies to retain copyright. For instance, non-profit organizations such as the FSF or the GNOME Foundation actively ask developers to assign the copyright to them. The reason for this is that these bodies may have stronger legal bodies to defend themselves from license infringement.

Therefore, it is very likely that if a source code file is owned by a company or one of these organizations, its copyright notice will appear in it. Usually, individual authors also include their copyright attribution, but they may be not that strict about that. In any case, the methodology of the study described in this paper is based on the assumption of the existence of those notices.

As the rest of the software industry does, copyright notices are included (among other places) at the beginning of each file with source code [18]. That means that information about the copyright holder can be extracted from source code files. For instance, the notice in the apps/units.c file of the GIMP project (see figure 1) clearly states that the copyright holders are Spencer Kimball and Peter Mattis, and that the license in use is the GNU General Public License. In any case, it should be noted that the way of stating the copyright is not unique and may change from project to project, even from file to file.

```
/* The GIMP -- an image manipulation program
 * Copyright (C) 1995 Spencer Kimball and Peter Mattis
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
[...]
```

Fig. 1. First lines of file apps/units.c, of the GIMP project

To identify those copyright notices, and extract from them information about the copyright owner, we designed the methodology described in the following subsections, and implemented it by producing pyTernity[3]. The structure

[3] The most current version of pyTernity can be found at `https://forge.morfeo-project.org/projects/libresoft-tools/`

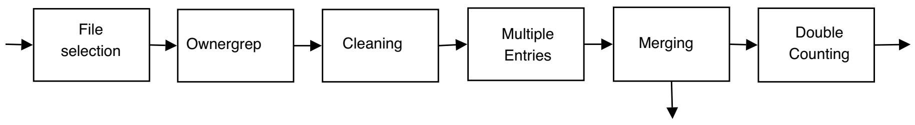


Fig. 2. Block diagram with the various components of pyTernity

of the methodology, which corresponds to the architecture of pyTernity, is shown in figure 2. The result is a list of files (avoiding similar files), with their size (in SLOC, lines, and characters) and the copyright holders identified in them.

2.1 File selection and counting

First step in the methodology is the identification of source code files. This is usually performed by using some ad-hoc heuristics, which may vary in their accuracy as well as in the granularity of their results. We use two sets of heuristics in our discrimination process: the extension in the file name, and the content. Detailed information about the process can be obtained from [13]. With these heuristics, almost all source code files are identified [15].

Every file identified as source code is then counted using SLOCCount[4], a tool authored by David Wheeler that calculates the number of SLOC (source lines of code). SLOCCount has been used in many studies about the size of software collections [21, 10, 1, 11, 14]. It calculates the number of physical source lines of code (SLOC) of a software program. The Unix wc command is also used to estimate the number of characters and lines of the file. All this information is stored in a database, linked to the file name.

2.2 Ownergrep

The second step is to search for copyright notices in source code files. For that, the *ownergrep* expression is compared with every line in the file. Since there is no standard or widely-used way of stating copyright in files, the pattern requires flexibility, which is achieved by the use of regular expressions that allow matching multiple, slightly different ways of expressing the copyright notice.

The *ownergrep* expression is a modification of the original one by Prakash and Ghosh (cite) and looks like this:

```
[1] .*copyright (?:\(c\))?[\d\,\-\s\:]+(?:by\s+)?([^\d]*)
```

Figure 3 presents some copyright entries that can be identified by the *ownergrep* expression. Identities found are stored in a database, linked to the files in which they were present (and therefore, also to their size).

[4] http://www.dwheeler.com/sloccount/

```
   Copyright (c) 1998, 1999 by Sun Microsystems, Inc. All Rights Reserved.
   Copyright (c) 2001-2, Vipul Ved Prakash.  All rights reserved.
   Copyright (c) 2006 IBM Corporation and others.
```

Fig. 3. Some copyright entries that can be matched by the *ownergrep* expression.

2.3 Cleaning

Identities stored in the database have to be cleaned. This means removing all non-relevant information to convert identified identities to their canonical format. This ranges from removing additional white-spaces to the deletion of dots. Some ad-hoc heuristics are used, along with the complement of a database of common transformations. Cleaning also includes splitting up an entry when it corresponds to two or more authors. So, the entry “Spencer Kimball and Peter Mattis” will result in two, one for Spencer Kimball and another one for Peter Mattis. If this is the case, both names appear as authors of the file and get attributed half of its length.

2.4 Multiple entries

After cleaning found identities, those corresponding to the same real entity are identified. Developers and companies may appear in several forms while corresponding to one single entity. The first idea in this line resulted in the construction of a large database where the various entries identified for a given developer were noted (manually). This method proved to enhance results considerably. However, the consideration of other methods, and the rising in complexity and size of the database have finally lead to the construction of a different tool, Seal, that returns a unique identifier for any given identity [16]. It is responsibility of this external tool to track all developers and to manage them properly.

2.5 Merging

Once cleaning has been performed and multiple entries have been identified, pyTernity merges the identities in the database so that authors appear only once in a file. This procedure also includes adding the size of all the files corresponding to each real identity.

2.6 Avoiding double-counting

In a large collection of software, some files may appear in several packages. That means that the copyright owner of one such file will be attributed the same code several times, which leads to inconsistencies. Therefore, similar files have to be identifies and removed from the count. For this, we use Nilsimsa[5], a hashing

[5] The Nilsimsa code can be retrieved from `http://ixazon.dynip.com/\%7ecmeclax/nilsimsa.html`

algorithm that produces similar hashes (according to a certain metric, based on Hamming distance) for similar texts. For our methodology, 32 bits of Hamming distance are used as the threshold for considering two files too similar to count them twice.

Unfortunately, comparing proximity of Nilsimsa hashes is meaningfully slower than comparing for equality. Therefore, comparing Nilsimsa hashes for every pair of files is not practical for large quantities of software. To avoid this situation, we propose a simplified use of Nilsimsa by (1) identifying files with the same Nilsimsa and similar amount of code in order to avoid false positives and (2) comparing by pairs all those files with the same filename. Table 1 shows the number of files with the same name and a similar nilsimsa hash that have been discovered for all versions of Debian.

Version	Total files	Same filename, similar Nilsimsa	Percentage
Debian 2.0	243,057	45,850	18.86%
Debian 2.1	367,463	80,551	21.92%
Debian 2.2	838,834	238,601	28.44%
Debian 3.0	1,340,081	292,367	21.82%
Debian 3.1	2,497,636	420,885	16.85%

Table 1. Total number of files and files that have the same file name and a similar Nilsimsa hash for every version under study.

2.7 Previous work

CODD, a tool designed by Rishab A. Ghosh and Vipul Prakash [8], was the first tool to extract authorship information from source code by tracking copyright notices. Its main aim is to assign the length (in bytes) of each file to the corresponding authors. It was successfully used in the Orbiten Survey [8], the source code survey in the FLOSS study [9], and some other research projects.

CODD is a very powerful tool which implements a methodology similar (in part) to pyTernity, but shows also some weaknesses. The most important one is that it lacks a way of merging the various ways in which an author may appear. So, authors may appear several times with different names or e-mail addresses. For instance, we have found that some developers have up to 15 different identities which may appear in copyright notices. In the case of companies and organizations, the same may happen: IBM or the MIT appear in several ways (up to twenty!) with slightly different wordings.

CODD also includes some heuristics for cleaning the extracted data. Although they have proven to be very powerful, these heuristics can not deal with enough accuracy with the fact that developers use different conventions to assign copyright.

Both limitations are important, and were the main reasons to create pyTernity. The *ownergrep* expression used in pyTernity is a modification of the original one in CODD.

3 Results on Debian

The methodology presented in the previous section has been applied to several releases of the Debian GNU/Linux distribution. In the next subsections an introduction to the Debian project and the results of our study are shown.

3.1 Introduction to Debian and global results

Debian is a libre operating system that, at present time, uses the Linux kernel to carry out its distribution (although there are some efforts to make that future Debian distributions could be based on other kernels). The distribution has a categorization of software packages according to their license and their distribution requirements. The main part of the Debian distribution (the section called *main*, which contains a large variety of packages) is compound only of libre software in agreement with the Debian Free Software Guidelines. It is available for download from the Internet and many resellers supply it on CDs or by other means.

The Debian distribution is created by over a thousand volunteers (generally computer professionals). The work of these volunteers consists on taking the source programs -in most of the cases from their original author(s)-, to configure them, to compile them and to pack them, so that a typical user of a Debian distribution only has to select the package to be installed/updated/removed. Hence, being a software included in Debian depends only on a volunteer performing the aforementioned tasks.

	Codename	Release	Packages	Total SLOC	Companies SLOC	%	# Companies
2.0	Hamm	Jul 1998	1,096	28,750,853	4,259,164	6.75%	249
2.1	Slink	Mar 1999	1,551	44,352,088	6,477,981	6.85%	312
2.2	Potato	Aug 2000	2,611	95,738,163	14,934,951	6.41%	482
3.0	Woody	Jul 2002	4,579	151,023,303	23,271,027	6.49%	782
3.1	Sarge	Jun 2005	8,560	239,580,490	40,421,751	5.93%	1455

Fig. 4. Some information about the Debian distributions under study: version number, *Toy Story* codename, release date, number of source code packages, total number of SLOC, SLOC attributed to companies, share of code by companies and number of different companies identified.

Table 4 gives further details about the various releases that have been studied in this paper and about the involvement of firms in them. As it is already known from previous studies [14] the size of Debian seems to double almost every two years. It is noteworthy that the amount of code that can be attributed to companies stays almost constant over time with values that lie around 6%-7%, throwing that the number of lines of code contributed by companies grows at the same pace than the distribution. The number of companies that could be identified has also increased significantly from over 200 in 1998 to almost 1500 in the most recent stable version. In any case, from our empirical analysis we can conclude that the involvement of industry in the libre software phenomenon has grown substantially in the last 8 years, although its relative importance has remained constant. It should be noted that this may not mean that the participation of the software industry has not been raising in the last years as other activities different from development such as support, consultancy and deployment without providing feedback to the community are not considered with our methodology.

3.2 Top companies by non-double-counted SLOC

Tables 5, 6 and 7 give the contribution of companies found in the various Debian releases under study. Contributions are measured in SLOC, avoiding double counting as explained in the methodology. These stats may give an idea of the effort spent by the company in the development of libre software.

Company name	SLOC	Files
sun microsystems inc.	801,632	2644
digital equipment corp.	434,152	1119
silicon graphics corp.	277,992	1274
xerox corp.	207,623	736
aladdin enterprises	92,172	475
age logic inc.	79,458	217
nec corp.	78,538	135
e.i. du pont de nemours	76,458	45
hewlett packard corp.	71,201	283
evans & sutherland	66,840	95

Company name	SLOC	Files
netscape comm. corp.	1,129,302	3934
sun microsystems inc.	810,437	2716
digital equipment corp.	428,176	1100
silicon graphics corp.	277,409	1207
aladdin enterprises	141,652	656
xerox corp.	98,071	342
lucent technologies inc.	85,586	139
at&t	80,140	223
age logic inc.	79,458	217
nec corp.	78,538	135

Fig. 5. Top-contributing companies for Debian 2.0 and Debian 2.1 (non-double-counted SLOC).

SUN Microsystems has historically been among the most contributing firms in terms of lines of code. For the first four Debian versions considered, its contribution was slightly less than one million lines of code, but with the inclusion of OpenOffice.org in Debian 3.1 its share has increased notably with over 5 MSLOC. IBM is another software *giant* present in this list, although its appearance is more recent (it enters the top 10 only in Debian 2.2). Interestingly enough, we find that the third place in Debian 3.1 is occupied by a company which is the main driver of a competing distribution to Debian, Red Hat Corp.

Company name	SLOC	Files
netscape comm. corp.	2,651,592	10423
sun microsystems inc.	1,086,765	4418
digital equipment corp.	975,178	5355
silicon graphics corp.	310,640	1308
aladdin enterprises	296,933	1403
ibm corp.	226,386	479
trolltech as	147,154	587
lucent technologies inc.	146,014	208
e.i. du pont de nemours	140,351	136
xerox corp.	128,922	444

Company name	SLOC	Files
ibm corp.	1,258,263	4832
sun microsystems inc.	955,462	3276
digital equipment corp.	784,279	4810
trolltech as	587,784	1836
silicon graphics corp.	575,810	2818
red hat corp.	376,099	878
static free software	292,448	242
aladdin enterprises	284,422	1422
abisource inc.	232,795	1530
hewlett packard corp.	208,903	707

Fig. 6. Top-contributing companies for Debian 2.2 and Debian 3.0 (non-double-counted SLOC).

Company name	SLOC	Files
sun microsystems inc.	6,025,680	22,720
ibm corp.	1,991,300	6,953
red hat corp.	1,366,298	4,807
silicon graphics corp.	1,111,431	4,422
sap ag	1,080,246	4548
mysql ab	852,394	2,425
netscape communications corp.	786,070	2,780
ximian inc.	750,761	2,924
realnetworks inc	673,167	2453
at&t	656,045	2,620

Fig. 7. Top-contributing companies for Debian 3.1 (non-double-counted SLOC).

This is due to its participation in projects such as GNOME or the GCC compiler collection[6].

4 Conclusions

In this paper we have described a methodology (and pyTernity, a tool implementing it) which can be used to scan source code files and find their copyright owners. This methodology is used to estimate (over time) the quantity of code owned by companies in the largest libre software collection: the Debian GNU/Linux.

The information resulting from this estimation is a first try with the aim of answering several questions about the presence of companies in libre software development. For instance, the share of code owned by companies has been calculated (being around 6%-7% for all the studied releases of Debian, with some tendency to lower), and the list of the companies contributing with more

[6] Cygnus Solutions Inc. was acquired by Red Hat in 1999.

code - which is leaded by giants like Sun Microsystems, IBM, SAP, Silicon Graphics or AT&T, but also includes more small, focused on libre software companies like Red Hat, Ximian (now owned by Novell) or MySQL.

The described methodology can provide this landscape of company involvement, but has to be considered with some care, since several sources of potential errors do exist. To begin with, it is completely based on the accurate identification of copyright notices, and correct extraction of identities from them. This is based in heuristics which, even having been validated in several ways, may not completely identify some copyright owners, or could wrongly assign code to others. In addition, the identification of multiple identities for a single identity, or the presence of several copies of some source code files could lead to miscalculations, which are dealt with by the methodology, but again using heuristics with a certain chance of error.

However, manual validation of a random set of results seem to lead to the conclusion that the results are good enough for using them to better understand how much code from companies can be found.

Several lines of research are still open in this area. First of all, further developments in heuristics to better identify companies and other institutions from copyright notices would improve the accuracy of results. In addition, improvements in the merging of different identifications corresponding to the same entity would also help.

Correlation of these data, and comparison with performance parameters of both the products and the companies with different levels of involvement in libre software development are also promising lines for interesting results. Methodologies to automatically assess companies and other entities about errors in copyright attributions, and about ownership of the code corresponding to software they use could also be of great interest to industry.

5 Acknowledgments

We would like to thank Diego Barceló and the rest of the GSyC/LibreSoft research group at the URJC for their invaluable help with the implementation and testing of the software used in this study. We are also very grateful to Rishab A. Ghosh, Rüdiger Glott and Kirsten Haaland from UNU-MERIT/University of Maastricht.

References

1. Juan José Amor, Jesús M. González-Barahona, Gregorio Robles, and Israel Herraiz. Measuring libre software using Debian 3.1 (sarge) as a case study: preliminary results. *Upgrade Magazine*, August 2005.
2. Andrea Bonaccorsi and Cristina Rossi. Comparing motivations of individual programmers and firms to take part in the open source movement. from community

to business. Technical report, University of Pisa; Sant'Anna School of Advanced Studies, Italy, 2003.

3. Andrea Bonaccorsi and Cristina Rossi. Altruistic individuals, selfish firms? the structure of motivation in open source software. *First Monday*, 1(9), January 2004.
4. Andrea Bonaccorsi and Cristina Rossi. Open source software, intrinsic motivations and profit-oriented firms. do not firms practise what they preach? In *Proceedings of the 1st International Conference on Open Source Systems*, Genoa, Italy, July 2005.
5. Andrea Bonaccorsi, Cristina Rossi, and Silvia Giannangeli. Adaptive entry strategies under dominant standards: Hybrid business models in the open source software industry. Technical report, University of Pisa; Sant'Anna School of Advanced Studies, Italy, 2003.
6. Kevin Crowston and James Howison. The social structure of open source software development teams. In *Proceedings of the International Conference on Information Systems*, Seattle, WA, USA, 2003.
7. Kevin Crowston and James Howison. The social structure of free and open source software development. *First Monday*, 10(2), February 2005.
8. Rishab A. Ghosh and Vipul Ved Prakash. The orbiten free software survey. *First Monday*, 5(7), May 2000.
9. Rishab Aiyer Ghosh, Gregorio Robles, and Ruediger Glott. Software source code survey (free/libre and open source software: Survey and study). Technical report, International Institute of Infonomics. University of Maastricht, The Netherlands, June 2002.
10. Jesús M. González-Barahona, Miguel A. Ortuño Pérez, Pedro de las Heras Quiros, José Centeno González, and Vicente Matellán Olivera. Counting potatoes: the size of Debian 2.2. *Upgrade Magazine*, II(6):60–66, December 2001.
11. Jesús M. González-Barahona, Gregorio Robles, Miguel Ortuño Pérez, Luis Rodero-Merino, José Centeno González, Vicente Matellan-Olivera, Eva Castro-Barbero, and Pedro de-las Heras-Quirós. Analyzing the anatomy of GNU/Linux distributions: methodology and case studies (Red Hat and Debian). In Stefan Koch, editor, *Free/Open Source Software Development*, pages 27–58. Idea Group Publishing, Hershey, Pennsylvania, USA, 2004.
12. Martin Michlmayr. Managing volunteer activity in free software projects. In *Proceedings of the USENIX 2004 Annual Technical Conference, FREENIX Track*, pages 93–102, Boston, USA, 2004.
13. Gregorio Robles. *Empirical Software Engineering Research on Libre Software: Data Sources, Methodologies and Results.* PhD thesis, Escuela Superior de Ciencias Experimentales y Tecnología, Universidad Rey Juan Carlos, 2006.
14. Gregorio Robles, Jesus M. Gonzalez-Barahona, Martin Michlmayr, and Juan Jose Amor. Mining large software compilations over time: Another perspective of software evolution. In *Proceedings of the Third International Workshop on Mining Software Repositories*, pages 3–9, Shanghai, China, May 2006.
15. Gregorio Robles, Jesus M. González-Barahona, and Juan-Julián Merelo. Beyond executable source code: The importance of other source artifacts in software development (a case study). *Journal of Systems and Software*, 80(9):1233–1248, September 2006.
16. Gregorio Robles and Jesús M. González-Barahona. Developer identification methods for integrated data from various sources. In *Proceedings of the International*

Workshop on Mining Software Repositories, pages 106–110, St. Louis, Missouri, USA, May 2005.

17. Gregorio Robles, Jesús M. González-Barahona, and Martin Michlmayr. Evolution of volunteer participation in libre software projects: evidence from Debian. In *Proceedings of the 1st International Conference on Open Source Systems*, pages 100–107, Genoa, Italy, July 2005.
18. Diomidis Spinellis. *Code Reading: The Open Source Perspective.* Addison Wesley Professional, 2003.
19. Ilkka Tuomi. Evolution of the Linux Credits file: Methodological challenges and reference data for Open Source research. *First Monday*, 9(6), June 2004.
20. Georg von Krogh, Sebastian Spaeth, and Karim R. Lakhani. Community, joining, and specialization in Open Source Software innovation: A case study. *MIT Sloan Working Paper No. 4413-03*, 2003.
21. David A. Wheeler. More than a gigabuck: Estimating GNU/Linux's size, June 2001.

Sprint-driven development: working, learning and the process of enculturation in the PyPy community

Anders Sigfridsson, Gabriela Avram, Anne Sheehan and Daniel K. Sullivan
Interaction Design Centre, Dept. of Computer Science & IS
Engineering Research Building, ER1002, University of Limerick, Ireland.
{anders.sigfridsson, gabriela.avram, anne.sheehan, daniel.sullivan} @ul.ie
WWW home page: http://www.idc.ul.ie

Abstract. In this paper we examine sprint-driven software development as it occurs in a specific Open Source community, PyPy. Applying a situated learning perspective, we report the findings from a study focused on the activities leading up to, taking place during, and following after sprints. The study included analyses of sprint reports, email archives and other documents available on the community website, as well as a one-week period of direct observation of a specific sprint. The objective of the study was to elaborate on how the practices of sprint-driven development in the PyPy community facilitate learning, the dissemination of knowledge among its members and the expansion of the Open Source community. This paper aims to assess how sprint-driven development can facilitate situated learning in distributed software development by describing the practices applied in PyPy.

Keywords: Distributed software development, Open Source communities, sprints, situated learning.

1 Introduction

Software development is a complex task. It is an activity which not only requires people with highly specialized technical skills, who are capable of working with highly abstract constructs and keeping up to date in an uncertain and rapidly developing area, but it also requires a high degree of collaboration. A software development project is often characterised by large scale, uncertainties, and complex interdependencies [14]. Further adding to these difficulties is the fact that software development is increasingly carried out in a distributed manner, fuelled by the complexity and large scale of modern software systems, by the trend toward globalization and the search for an educated yet inexpensive work force.

While the challenges facing globally distributed software development are not unique, certain difficulties – technical and managerial, as well as social and cultural - are further exacerbated by geographical and temporal distance. Three often mentioned issues are cultural differences, trust, and communication [e.g. 10, 19, 22]. But these are by no means the only difficulties, as more traditional concerns such as coordination, control and software processes are also affected by the distribution [e.g. 4, 11]. Currently there is significant interest in both academia and industry to gain a better understanding of these key issues and, above all, to discover ways of addressing the difficulties and thus improving practice [9].

A number of the most complex and successful software products nowadays – Linux, Apache, Firefox, OpenOffice and Eclipse, to mention but a few – have been developed or enhanced by Open Source Communities. The growing success of Open Source Software has resulted in it becoming a focus for research into issues relating to distributed software development. What is interesting about this phenomenon is how these loosely organized and often ad-hoc communities, using mostly simple communication and development tools such as email lists, version control systems and simple text editors [e.g. 8], can manage to develop high quality software [e.g. 17, 13].

Whilst we must recognize that the practices of Open Source communities are by no means the "silver bullet" for developing software and that many of them may not be adaptable to the more rigid requirements of the corporate world, they still provide a valuable resource in terms of understanding the key issues relating to distributed software development thus potentially providing guidance in the improvement of practice. Within the Open Source arena it is quite common that novel development methods and ways of working in cooperative projects are tried out. Some projects not only aim at producing an operational end product, but also actively investigate and improve software development techniques and attempt to find improved ways of running software development projects. One such Open Source project is PyPy.

The PyPy project[1] evolved from within the Python Open Source community and is focused on re-implementing the Python programming language using Python itself. The end-product will be an open run-time environment for the Python language, but this is not the only goal. It also focuses on investigating novel techniques for implementing practical dynamic programming languages and aims to showcase a software development method called "sprint-driven development". In this paper, we are focusing on this latter aspect.

We have conducted a study of the activities in PyPy consisting of document analysis of mailing lists, archives, sprint reports and other documents available on the community website[2] as well as a direct observation of one PyPy sprint, which took place in August 2006 in Limerick, Ireland. The objective of the study was to examine the actual activities leading up to, taking place during and following after sprints and to elaborate on how sprint-driven development facilitates learning, the dissemination of knowledge among its members and the expansion of the Open Source community. This paper will present a brief introduction to the PyPy project and the principles of sprint-driven development, and will then provide some specific

[1] http://pypy.org/
[2] http://codespeak.net/

accounts of the collaborative practices that occur in this community. We will apply a situated learning perspective to explain what we have observed and will draw conclusions about what lessons can be learned from PyPy regarding how sprint-driven development facilitates situated learning in distributed software development.

1.1 The socGSD project

At the University of Limerick, Ireland, a group at the Interaction Design Centre has received national funding as part of a software engineering research consortium to study the social, organisational, and cultural aspects of global software development (socGSD). The socGSD project aims to explore through case studies, how organizations attempt to manage the coordination of engineering work via a variety of mechanisms, from the formation of closely-knit, though distributed, teams in multinational companies through to Open Source communities, who act as self-organising bodies and manage to produce notable results without having formal management structures and too many well-defined rules. Our research is based on the findings of earlier studies on articulation and coordination work, information sharing, knowledge management and informal learning practices in distributed work. Our work is exploring the diversity of ways in which distributed teams shape their work practices and come to a joint understanding of their objectives. Our research also considers the various ways in which developers acquire new skills through their day-to-day practice and continuously improve their practice through learning and innovation.

Our research methods mainly rely on an interpretive, naturalistic approach to data collection and analysis. This means that we study the phenomenon in the actual settings where the work activity takes place, attempting to make sense of the work through the eyes of those actually doing it.

The study of an Open Source community for the duration of a sprint provided us with an excellent opportunity to observe the actual work practices of a team of developers who were collocated for one week, but also to consider these practices from the perspective of the context offered by the community's web presence and accounts of similar events.

1.2 Situated learning

Various theories of learning exist, each emphasizing different aspects of learning and embracing different fundamental assumptions regarding the nature of knowledge, learning and the role of the individual learner [15, 3, 23]. According to Lave&Wenger [15], situated learning can be considered as a bridge between a view according to which cognitive processes are primary and a view according to which social practice is the primary, generative phenomenon and learning is one of its characteristics. From this latter perspective, learning is viewed as an integral and inseparable aspect of social practice. Our study adopts a social practice theory of learning. In particular we are influenced by the concepts of situated learning, specifically legitimate peripheral participation, an analytical perspective introduced by Lave & Wenger [15] as a way of understanding learning.

According to Lave & Wenger [15] knowledge is learned by becoming a legitimate peripheral participant in a community of practice and by gradually acquiring "mastery", or knowledge and reputation, through a process of social interaction. Learning is thus not the result of direct and intentional teaching; rather it is enabled by participation in practice and access to the learning resources available in the community [24]. Active participation of newcomers allows them to interact with more knowleageable peers and provides access to the expertise available within the community. Learners acquire not just formal knowledge and skill, but also the ability to behave as members of a particular community of practice. In the words of Brown & Duguid [3] it involves becoming an "insider" or "*becoming* a practitioner not learning *about* practice." This situates learning squarely in the practices and communities in which the knowledge takes on meaning and significance.

Both Orr's [18] and Lave & Wenger's [15] research emphasizes that to understand working and learning, it is necessary to focus on the formation and change of the communities in which work takes place. Based on his ethnographic research on photocopier repair technicians, Orr posits that "not only is learning in this case inseparable from working, but also individual learning is inseparable from collective learning." The implication is that knowledge and learning are not simply the property of the individual, but are socially constructed and distributed. Hence what is learned is connected to the context in which it is learned and so learning can be fostered by fostering access to and membership of a particular community of practice.

The application of a situated learning perspective to distributed teams and Open Source Communities is not new [e.g. 24, 8, 21]. According to Ye & Kishida [24], an Open Source community requires a high degree of openness in terms of both process and product, as it offers more learning resources to encourage participation. In addition, the manner in which a software system is partitioned also has an impact on knowledge acquisition. By allowing newcomers to work on relatively independent tasks, each with progressive difficulty, it fosters the possibility of legitimate peripheral participation. In other words, it allows newcomers to participate peripherally by contributing to tasks at their current skill level and to gradually move on to take charge of more difficult tasks as mastery evolves. Furthermore, research by Gutwin et al. [8] on awareness in distributed software development highlights the importance of facilitating peripheral participation through email and chat. The mechanism of "overhearing" inherent in these text-based communication tools allows developers to become peripheral participants in each others conversations, thus providing valuable awareness and enabling "expertise" to gradually become visible.

2 The PyPy project and sprint-driven development

2.1 The PyPy project

PyPy is part of the large Open Source community behind Python. Python is a programming language, published under an OSI approved Open Source License. The Python language was originally developed in 1990 by Guido van Rossum. Today, the *de facto* standard implementation of the language is the CPython implementation, which is also being developed as an Open Source project

The PyPy project also aims at producing an implementation of Python. But unlike CPython, which is developed and written in C, the PyPy project is developing an interpreter for the Python language in Python itself (hence the project name).[3]

However, creating a run-time environment for Python is not the only purpose of this project. The PyPy project came into being as an Open Source project in 2003 and in December 2004 the project received partial funding from the European Union (EU). As a result, the project objectives expanded to include a methodological goal, namely to demonstrate that the Open Source way of working in general, and the development methodology of choice in particular, are successful ways of undertaking distributed, collaborative work and hence can be of use in future EU projects as well as in large-scale development projects in general. The methodology adopted by the PyPy community is what has been called “sprint driven development”.

2.2 Sprint-driven development

A “sprint” is a focused development session – developers gather in one place for a short period of time and work in pairs (or small groups) on specific parts of the software system. This type of event has become popular within some Open Source communities – for example, the OpenBSD and Linux communities - and has many names, such as “hackathon”, “codefest”, “sprintathon”, “sprint”, and so on. The primary purpose of these on-site meetings, which last from a few days up to one week, is to write and test code in a collaborative way. To facilitate access, these events are often collocated with conferences of relevance to the community’s members, but they may also be hosted separately in various locations, usually organized by community members or hosted by sponsors.

The practice of using sprints for pivotal development was initiated by the Zope Corporation in the early days of the Zope 3 project[4]. In order to maintain focus, the traditional sprint is supposed to last for only three to four days and to involve no more than 10 people. A sprint generally incorporates aspects of eXtreme Programming such as pair-programming and test-driven development. In addition, it is usually led by a “coach”, who sets the goals, organizes the event, coordinates the work, tracks the results and follows up.

The underlying concept is that a sprint is a good way to give a project “a boost by focusing the efforts of a group on specific development issues” [12]. Furthermore, sprints also offer valuable opportunities to maintain developer involvement, and to enable newcomers to get acquainted with the code base as well as the specifics of a project.

[3] For technical specifications, http://codespeak.net/pypy/dist/pypy/doc/architecture.html

[4] http://www.zopemag.com/Guides/miniGuide_ZopeSprinting.html

2.3 Sprint-driven development in practice in PyPy

The PyPy community describe themselves as a hybrid project, combining different aspects of Agile and Distributed Development within the context of an Open Source community [5]. In PyPy the developers are not just distributed but also dispersed, with no more than a few developers being located in the same place. The main strategy in PyPy to handle this challenge to the development process is to "sprint" systematically, using sprints not only for software iteration purposes but also to provide an accelerated and collaborative physical practice that enables community building as well as the dissemination of knowledge and learning within the team. In fact the PyPy project itself originated from a one-week sprint held in February 2003.

The sprint methodology used with the PyPy community differs in a number of ways from the original Zope3 format described earlier. The focus of Zope3 sprints was to produce code and as such they tended to be rather closed events where only experienced Zope developers participated and they were usually arranged close to larger releases [5]. In addition, an appointed "coach" was used to coordinate the event and its outcome. However, within the PyPy community a sprint is an open event where newcomers are welcomed – indeed a sprint is seen as an opportunity to initiate newcomers into the project and clearly has a "tutorial" component. In addition, PyPy sprints are developer-driven and no formal role such as a "coach" exists. Instead, they have introduced a mechanism of initial and daily status meetings where the whole group makes decisions. A local contact will help to organize the logistics for the event based on the sprint location.

A PyPy sprint is usually 7 days long, with one free day in the middle normally dedicated to social events. The sprint is initiated with a start-up meeting. Tutorials will be arranged during the sprint if there are new participants present or if a new tool or feature has been implemented. For the remainder of the week, each day begins with a status meeting. During the status meetings, progress is discussed, tasks are drafted, the direction of the sprint is set or altered, and developers pair up according to needs, skills and wishes. During sprints pair-programming is used systematically – not only between core developers sharing an interest in a specific task but also for mentoring newcomers by pairing them with core developers [5]. The pairs may change each day, or may continue to work together for several days. Apart from the actual code, the outcome of a sprint is also a sprint report. The sprint report summarizes the activities and the initial goals and results. It also serves as an orientation for focusing the work of the community until the next sprint.

In PyPy, there is a rough plan detailing future sprints for the coming months, enough to maintain a general awareness of the dates and sites of upcoming sprints and allowing people to plan for attendance. About a month before a particular sprint, its content and goals are discussed on the mailing list (pypy-dev) and on the PyPy IRC channel, mostly by the core developers, although the discussions are transparent and anyone can, in principle, participate.[5] Information is also distributed on the general pypy-sprint mailing list and through the project webpage. As the sprint approaches, a more detailed sprint announcement is sent out. People can announce their intention to attend either by checking in the information in Subversion (the

[5] http://codespeak.net/pypy/dist/pypy/doc/dev_method

PyPy code repository and version control system), or by posting on the sprint mailing list. Lately, the PyPy project introduced “pypy-sync meetings” (on IRC) and this has also become a major forum for discussing the content and goals of upcoming sprints where any member can participate.

3 Research Method

In our study of collaborative work practices, the preferred methods are inspired by ethnography. We try, whenever possible, to observe people in their normal work environment as they engage with their work practice. Furthermore, we interview the participants (individually or in groups) and bring into discussion events we observed, complementing what we saw with the addition of their perspectives. An ethnographic approach typically includes field work done in natural settings, the study of the larger picture to provide a more complete context of activity, an objective perspective with rich descriptions of people, environments and interactions, and an aim toward understanding activities from the informants’ perspective [1]. More recent studies [16] claim that by narrowing the focus of field research before entering the field, using key informants and multiple interactive observation techniques and collaborative iterative data analysis methods, one can obtain reliable data in a shorter period of time than was traditionally considered.

The study we conducted was mainly centred on the sprint that took place in Limerick, Ireland, between the 21st and 28th of August, 2006. The sprint was hosted at the University of Limerick, with the assistance of local contacts. Three researchers where involved in this observational study, but none participated actively in the coding efforts. For the most part, there were 7 participants in this sprint, mostly core developers. A local developer joined the sprint for the last three days, and two newcomers also visited and attended a tutorial that was arranged for them. Since there were mostly core developers present, the sprint was considered an opportunity to work on some of the more crucial technical matters, e.g. the JIT module, core optimization and distributed testing.[6]

We studied this sprint mostly through direct observation, complemented by informal discussion and a dedicated Q&A session. We observed and recorded (video and audio) the start-up meeting and the daily status meetings, as well as observed some of the actual work sessions. Because of the interest expressed by two different groups of researchers at the University of Limerick in the PyPy way of working, the project manager organized a workshop on the first day of the sprint where the PyPy project and the sprint-driven methodology was presented. One of the most senior members of the project joined the last half of the workshop and there was a Q&A session.

[6] http://codespeak.net/svn/pypy/extradoc/sprintinfo/ireland-2006/limerick_sprint-report.txt

Prior to the sprint, we reviewed a number of sources referring to Open Source communities in general and sprints, the Python language and the PyPy project in particular, including papers and talks, as well as mailing lists, web pages, bios, sprint reports, blog posts referring to PyPy, and so on. In order to get an insight into the activities of the project and the dynamics of the community, since it's inception, we studied the PyPy community's extensive online documentation (such as project descriptions and both sprint and EU reports), as well as mailing list archives and chat transcripts that are available on the website. After the sprint, we continued to observe the community for an additional four months, mostly through continued document analysis of email lists, sprint reports and other documents.

4 Sprints as a way of working, learning and innovating

Several authors speak about the various roles assumed by the members of Open Source communities [20, 24, 7]. The traditional evolution based on perceived levels of expertise, is from the periphery of the community to the centre: the majority of people start as users, get involved by discovering and later fixing some bugs, make occasional contributions to the source code, and only after gaining a reputation as an "expert" can they be accepted as core members of the community. The apprentice often has a long (and sometimes lonely) way to go before becoming actively involved in development. The PyPy community is, in this respect, quite different. There is no single leader or visionary – just a core group of passionate Python developers. Anyone who has the skills and motivation can rapidly become an active contributor, because within the PyPy community there is a welcoming attitude toward new participants which originates in the strong belief of the community members regarding the benefits of collaborative work. There is a strong culture of openness and transparency, or as described in [8] a culture of "keeping it public". Access to the PyPy online mailing lists and IRC is freely available. Core developers are accessible to answer questions or act as mentors both virtually via mailing lists and IRC and in person during sprints. The fact that the PyPy development process incorporates an automated framework for testing and version control allows for a more relaxed attitude regarding distribution of commit rights to new developers [5].

Several studies on Free and Open Source Software mention learning as one of the core motivations for participation [24, 7, 13], but in many cases, this simply means "lurking" and using the available code. While "lurking" - or in effect being a peripheral participant in the community - can provide valuable awareness information [8], in PyPy newcomers are encouraged to become directly involved in development from the very beginning. The PyPy community has developed a comprehensive and detailed repository of documentation, guides for beginners, talks, sprint reports, mail and chat archives in addition to its main code repository. While an important part of the PyPy community's body of knowledge is freely available on the web, becoming a member of the community is made quicker and easier by participation in collocated events such as sprints. Newcomers can make a decision about staying or leaving after being offered an immersion in the practices, social events and personal contacts that usually arise in a sprint.

Before deciding to join their first sprint, newcomers are encouraged to get accustomed to the work being done in PyPy. The architecture of the interpreter, the code itself, extensive coding guidelines, the available documentation, the tools used, configuration and various tutorials are all available on the PyPy website. Furthermore, newcomers are also encouraged to start socializing with the others by participating in email and IRC conversations and accessing the mail and chat archives. For example, the following excerpts from the PyPy mailing lists show how the community greets newcomers:

"Cool! Contributions are of course very welcome! I guess the most immediate step would be to read through the documentation and ask any question you might have (here – on the mailing list- or on the IRC channel). It certainly won't be a problem finding work for you :-)"

"In addition, note that this sprint is […] a coding sprint, and we specifically welcome newcomers. If possible and interesting for you, feel invited :-) That's the best way to grasp the basics of PyPy and discuss. Also feel free to say hello in the #pypy IRC channel (irc.freenode.net) and discuss your interests."

Subsequently, during the actual sprint, newcomers are given tutorials and then "taken by the hand" usually by pairing up with an experienced developer, working together on a chosen topic and getting detailed feedback.

The participants in the Limerick sprint in August 2006 were in the majority part of the core PyPy group, with one exception: a young and enthusiastic developer who was funded through the "Summer of PyPy" initiative[7] (although it was not his first PyPy sprint). Pairs were formed and topics were chosen in an extremely flexible way. The start-up meeting highlighted the list of topics that needed attention resulting from previous sprints and discussions, the participants announced their intentions to the group, paired up according to these, and simply started working on them. Although the project manager (who has an administrative role and is not involved in coding) and one of the core developers chaired the meeting, their role was more one of facilitating the sprint, and not imposing anything on the group. At the end of the week, this role was taken over by another member of the core group. Every decision was taken collectively, and the initial program changed several times to accommodate people and events. Usually, there's a day dedicated to social activities in the middle of the sprint week, but this time the group decided to continue working through the dedicated break day because of a slow start on Monday morning, and to have a night out on Friday instead, when the local developers were planning to join.

This is one illustration of how flexible the working style of PyPy sprints are and it shows that agility and the ability to incorporate continuous change and adaptation are highly valued by the PyPy community. They innovate continuously, looking for both solutions to make their software more efficient, and for practices that would allow them to enhance or improve the form but keep the spirit of their activities.

[7] http://codespeak.net/pypy/dist/pypy/doc/summer-of-pypy.html

The lack of formality and the relaxed atmosphere are probably the first striking aspect when observing a sprint. During the Limerick sprint, participants spoke to each other, moved around asking questions, joked and had fun. They were all located in the same room and maintained a certain awareness of what was happening in the other coding groups. They made the decision to take a break – or continue an hour more than planned – by consulting each other. Peers were invited to have a look when unexpected errors occurred or a new solution was tried out. Priorities were permanently shuffled, concepts re-invented, new routes adopted, tried out and sometimes abandoned.

In the Limerick sprint, different working styles could be observed in the pairs. In the first pair, one of the participants distributed his attention, switching between multiple windows, reading through his emails or keeping an eye on the chat channel while listening to a new solution proposed by his team-mate. His (more experienced) companion explained every step he was taking, made his reasoning transparent and asked a lot of provoking questions. A dialogue went on throughout the session: when the first developer had an idea he preferred to try it out instead of explaining it, while his colleague watched the screen, waiting to see the result. The second pair did not display as much interaction, perhaps because the tasks were divided more clearly between them. They seemed to work independently each on his own laptop, showing each other errors or successes and exchanging ideas only once in a while. The third pair was sharing a laptop. Most of the time, the laptop's owner was the one using it, but his actions seemed to result from their joint discussion. The conversation was vivid and emotional, accompanied by a lot of gestures.

The participants in a PyPy sprint benefit not only from their mutual knowledge sharing, but there's also a recognisable flow of enthusiasm. When speaking about the core group of developers during the methodology workshop, the project manager described them as "soulmates", who have much stronger bonds than the current EU project framework and want to continue working together after the end of this project. Sprints provide the opportunity for a process of learning and enculturation, where new participants get the chance to become directly involved not only in problem solving, innovation and planning, but also in the social life of the community.

5 Discussion

5.1 Learning facilitated by sprint-driven development

A major issue in distributed software development projects is how to facilitate learning about programming techniques, technology and project specific matters among project participants when direct interaction is limited due to geographical and temporal distance and, often, affected by national, social and organisational cultural differences. A sprint offers a good opportunity for the dissemination of knowledge, both among senior members of the community and to new members. However, being able to contribute to a software development project does not just require technical skills.

From a situated learning perspective, learning cannot be seen as an isolated activity, separated from the practice it is meant to enable [3]. Instead, learning involves becoming an “insider”, not just absorbing a discreet body of individual knowledge, but learning to function within the community of practice. So learning the necessary skills needed to participate in a project like PyPy also involves learning about the dynamics of the community, what norms and interpretive schemes are dominating, and what range of behaviour is acceptable, as well as developing an identity in the community.

PyPy sprints are a perfect illustration of situated learning, as conceptualized by Lave & Wenger [15]: newcomers begin by reading the information online and joining mailing lists and IRC channels and then eventually join their first sprint and get more and more involved in the general development effort, learning happens in a community of practice, by participation (a peripheral one in the beginning), and by gradually acquiring knowledge and reputation through social interaction. Brown & Duguid write that the “central issue in learning is *becoming* a practitioner not learning *about* practice” [3]. From what we have seen during this study, this is precisely what the PyPy people are supporting when welcoming new participants to sprints, arranging tutorials for them, and pairing them up with more experienced developers to do the work. This mechanism is further enhanced because the new participants are encouraged to participate in the mailing lists and IRC channels and to get acquainted with the system architecture and the code base prior to their first sprint, and thus have already started to form an identity within the community when arriving at the sprint.

Learning the concepts of the Python programming language does not mean one knows how to program in that language. Applying those concepts to a specific project and actually writing code is when learning happens. Sprints accelerate this process for distributed teams, recognising the important situated aspects of learning and supporting them.

5.2 Sprints as a way of sustaining and renewing the community

Previous research on Open Source software development has shown that learning is, in fact, a major motivational force for participants [24, 13]. It has also been argued that for Open Source projects to sustain themselves, the community must co-evolve with the system developed [24]. The community must be able to regenerate itself through both concrete contributions of code and the emergence of new members who can carry on the work. The sprints in PyPy, through conscious mentoring efforts, attract new members and enable them to both achieve the necessary technical skill and to create an identity within the community, thus ensuring the sustainability of the community.

However, regarding the formation of the community, there are also possible hazards with driving development through sprints. During the sprint, the centre-periphery relationship, usually based on experience and contributions resulting in a hierarchy in most Open Source communities, is altered: the collocated participants become the centre, while all the others move, in a way, to the periphery because they are missing from that specific location. The danger is that this leads to the formation of in-groups. The active PyPy coding effort is a subgroup within the wider Python community and those who participate in sprints are again a further subgroup (although temporary) of the overall coding effort. This situation can lend itself to the formation of in-groups and the exclusion of others and the eventually fragmentation of the group.

A previous experimental effort [2] to consider the in-group/out-group effect was concerned with the mixed media working environment whereby access to resources is not equally distributed. In part this is a consequence of having a substantial component of the development team collocated. The hypothesis which they examined was that individuals collocated together will interact more and form an in-group. We heard this concern voiced by the project manager herself. Since the progress is so rapid and so much happens during a sprint, they are aware that there is a risk that the non-participants can't keep up and can become passive. For example, this is acknowledged in one of the EU reports[8] where it is stated that, "due to the projects fast pace and its many developments, it requires substantial effort for the average community member to contribute to the project." However, in the PyPy project, there is conscious effort to ensure the community doesn't fragment and so "the mentoring and supporting activities from the EU project members have increased accordingly."

The strategy has been to host sprints at different locations to encourage and facilitate participation from as wide a group as possible. During the period 2003-2004 6 sprints were arranged in various European cities (since then there has been a more systematic structuring of sprinting every 6th week) [6]. Sprints have also been organized on other continents whenever possible. For example, there was a post-PyCon PyPy sprint in February 2006 in Dallas, USA, and another one in Tokyo, Japan in April 2006. Also, during the recent Leysin sprint, in January 2007, a remote participant worked constantly with two others participating in the sprint to accomplish a specific task. Non-European developers whose participation in sprints is more difficult to organise have raised the possibility of doing a "virtual sprint" that would enable them to get involved as well.

6 Conclusions

Our study has focused on the actual activities leading up to, taking place during and following after sprints and the purpose has been to elaborate on how sprint driven development facilitates learning, the dissemination of knowledge among its members and the expansion of the Open Source community. The aim of this paper has been to illustrate how sprint-driven development can facilitate situated learning in distributed software development by describing the practices applied in PyPy.

The observations indicate that the sprint-driven development methodology as it occurs in PyPy is interesting because, while it is a way to accelerate the development in terms of written code, it also serves as a mechanism to expand the community and

[8] http://codespeak.net/pypy/extradoc/eu-report/D14.3_Report_about_Milestone_Phase_2-final-2006-08-03.pdf

facilitate the enculturation of its members. In PyPy, we have seen how new participants are welcomed to sprints and how a real effort is made to include them in the community by encouraging participation in the online activities prior to their first sprint and arranging tutorials and pairing them up with experienced developers to work during the sprint. This attracts new members and enables them to both achieve the necessary technical skill and to create an identity within the community, thus enabling them to contribute. It also contributes to sustaining and renewing the PyPy community through the inclusion of new participants and the emergence of new core members and active developers.

7 Acknowledgements

The authors would like to extend thanks to the PyPy community and to the participants of the Limerick sprint for allowing us access and for being so accommodating. This work is part of the socGSD project at the University of Limerick. socGSD is one of the LERO (the Irish Software Engineering Research Institute) cluster projects funded under PI grant 03/IN3/1408C by the Science Foundation of Ireland (SFI).

8 References

1. Blomberg J. et al. (1993) Ethnographic field methods and the relation to design. In D. Schuler and A. Namioka, (Eds.) *Participatory Design*, Lawrence Erlbaum, pp. 123-155.
2. Bos N., N. S. Shami, et al. (2004). In groups/Out Group Effect in Distributed Teams: An Experimental Simulation. *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, Chicago, Illinois, USA, pp. 429-436.
3. Brown J. S. & Duguid P. (1991) Organizational learning and communities-of-practice: towards a unified view of working, learning and innovation. *Organization Science*, vol. 2, no. 1, Special Issue: Organizational Learning: Papers in Honour of (and by) James G. March (1991), pp. 40-57.
4. Carmel E. & P. Tjia (2005) *Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce*. Cambridge University Press, Cambridge, MA.
5. Düring B. (2006A) Sprint Driven Development: Agile Methodologies in a Distributed Open Source Project (PyPy). *The 7th International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Oulu, Finland.
6. Düring, B. (2006B). Trouble in Paradise: the Open Source Project PyPy, EU-Funding and Agile Practices. *AGILE 2006* Minneapolis, Minnesota, USA IEEE Computer Society's Digital Library.
7. Ghosh, R. A. & R. Glott (2002). Free/Libre and Open Source Software: Survey and Study- summary report. *Workshop on Advancing the Research Agenda on Free/Open Source Software*. Maastricht, Int'l Institute of Infonomics, Univ. of Maastricht.

8. Gutwin C., Penner R. & Schneider K. (2004) Group awareness in distributed software development. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pp. 72 – 81, 2004.

9. Hargreaves E., Damian D., Lanubile F. & Chisan J. (2004) Global Software Development: Building a Research Community. In *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 5, September 2004, pp. 1-5.

10. Herbsleb J. D. & Moitra D. (2001) Global Software Development. *IEEE SOFTWARE*, March/April 2001, pp. 16-20.

11. Herbsleb J. D. & Grinter R. E. (1999 A) Architectures, Coordination, and Distance: Conway's Law and Beyond. *IEEE Software*, 16(5): 63-70

12. Holden H. (2006) *Running a Sprint*. ONLamp.com, Python Development Center. Available:

    http://www.onlamp.com/pub/a/python/2006/10/19/running-a-sprint.html (15/01/07).

13. Kim E. E. (2003) *An Introduction to Open-Source communities*. Blue Oxen Associates. Available: www.blueoxen.com/research/00007/BOA-00007.pdf (19/01/07).

14. Kraut R. E. & Streeter L. A. (1995) Coordination in Software Development. In *Communications of the ACM*, vol. 38, no. 3, March 1995, pp. 69-81.

15. Lave J. & Wenger E. (1991) *Situated Learning: Legitimate Peripheral Participation*. New York: Cambridge University Press.

16. Millen D. R. (2000) Rapid ethnography: time deepening strategies for HCI field research. *Conference on Designing interactive systems: processes, practices, methods, and technique*, New York, ACM Press.

17. Mockus A., Fielding R. T. & Herbsleb J. D. (2002) Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 11, Issue 3 (July 2002) , pp. 309 – 346.

18. Orr J. (1996) *Talking about machines: An Ethnography of a Modern Job*, Ithaca, New York, IRL Press.

19. Prikladnicki R. et al. (2003) Global software development in practice: lessons learned. *Software process improvement and practice*, vol. 8, 267-281, 2003.

20. Rahtz, S. (2004) Building Open Source Communities, OSS Watch, University of Oxford, Available: http://www.oss-watch.ac.uk/talks/2004-11-19-bodington/ (20/12/06).

21. Robey D, Huoy Min Khoo & Powers, C. (2000) Situated Learning in Cross-functional Virtual Teams. *IEEE Transactions on Professional Communication*, Vol. 43, Issue 1, pp 51-66.

22. Sahay S., Nicholson B. & Krischna S. (2003) *Global IT Outsourcing: Software Development Across Borders*. Cambridge, UK: Cambridge University Press.

23. Wenger E. (1998) *Communitites of Practice: Learning, Meaning, and Identity*, Cambridge University Press.

24. Ye, Y. and K. Kishida (2003). Toward an Understanding of the Motivation of Open Source Software Developers. *International Conference on Software Engineering - ICSE2003*, Portland, OR.

Using Repository of Repositories (RoRs) to Study the Growth of F/OSS Projects: A Meta-Analysis Research Approach

Sulayman K. Sowe, L. Angelis, I. Stamelos, and Y. Manolopoulos

Department of Informatics, Aristotle University, 54124 Thessaloniki, Greece.
Tel: +30-2310-991927 Fax: +30-2310-998419
[*] `sksowe,lef,stamelos,manolopo{@csd.auth.gr}`

Abstract. Free/Open Source Software (F/OSS) repositories contain valuable data and their usefulness in studying software development and community activities continues to attract a lot of research attention. A trend in F/OSS studies is the use of metadata stored in a repository of repositories or RoRs. This paper utilizes data obtained from such RoRs -FLOSSmole- to study the types of projects being developed by the F/OSS community. We downloaded projects by topics data in five areas (Database, Internet, Software Development, Communications, and Games/Entertainment) from Flossmole's raw and summary data of the sourceforge repository. Time series analysis show the numbers of projects in the five topics are growing linearly. Further analysis supports our hypothesis that F/OSS development is moving "up the stack" from developer tools and infrastructure support to end-user applications such as Databases. The findings have implications for the interpretation of the F/OSS landscape, the utilization and adoption of open source databases, and problems researchers might face in obtaining and using data from RoRs.

Key words: Open Source Software Repositories, Metadata, Time Series Analysis, Missing Values Estimation, Projects Growth, Linear Trend, Open Source Databases.

1 Introduction

The user of Free and Open Source Software (F/OSS), having access to the source code, is free to study what the program does, modify it to suit his/her needs, distribute copies to other people and publish improved versions so that the whole F/OSS community can benefit. The licenses agreement (e.g. the General Public License or GPL) under which the source code is distributed defines exactly the rights the user has over the product. The Bazaar model [19] of developing F/OSS represents a significant shift in the way we develop and maintain traditional or closed-source software (CSS). As [27] pointed out, F/OSS development

[*] Correspondence author: Sulayman K Sowe; Email: sksowe@csd.auth.gr

differs in many ways from that of CSS, where it is common to assume centralized software development and administrative authority that controls and manages the resources and schedules for software development and maintenance. The model has produced a number of successful applications in the area of operating systems (Linux), emailing and web services (Gmail, Apache), databases (MySQL, PostgreSQL), to mention a few. Participants in F/OSS project rely on extensive peer collaboration through the Internet, using project's mailing lists, *de facto* versioning systems such as Concurrent Versions System (CVS) or Subversion (SVN), bug-tracking systems (BTS) and bug databases (e.g. Bugzilla), Internet Relay Chats (IRC), discussion forums, etc. These tools not only enable participants to collaborate in the software development process but also act as repositories to store the communication activities of the participants.

With the coming of F/OSS, sprang various portals to provide hosting services for projects of all kinds and flaviours. Among the largest and most popular is sourceforge. Freshmeat and Savannah also continue to attract a lot of attention. These portals are hosts to small and large, successful [7] and unsuccessful projects. Yet, many portals are also graveyards strewed with abandoned projects. The plethora of projects or applications available throughout the Internet is an indication of a growing interest in F/OSS and the fact that an increasing number of skilled programmers are willing to transform their [tacit] knowledge and skills into tangible products [24]. While corporations posting projects to sourceforge view their strategies as important in accelerating adoption and migration of their products and services [3], the F/OSS landscape is biased towards certain kinds of projects and software. What is more interesting about the landscape is that applications developed by the community are not uniformly distributed across all domains. Projects cover wide ranging topics, with development being dominated by infrastructure support or Internet based projects [22]. These products may fall under operating systems (Linux, FreeBSD), server and Internet applications (Apache, Sendmail, BIND), and software development tools (GCC, Perl, Python). Many of these projects resulted from an individual (usually a skilled programmer) scratching his own itch [19]. Ideally, if the required software is not freely available, one can either develop it on his own or contribute the initial code-base and release it to the F/OSS community for collaboration. Bezroukov, [4] noted that F/OSS projects are more successful in areas that are directly or indirectly interesting to developers themselves. However, itch-based F/OSS may not succeed in improving ease of use for those users, such as novices, whose background differs from professional developers [16]. Even though there are a growing number of applications targeting end-users (the KDE and GNOME desktops, Firefox, Thunderbird), our aim in this study is to discuss trends in F/OSS research and provide a quantitative analysis of the F/OSS landscape to test one hypothesis:

Hypothesis: *Is there evidence to support that F/OSS development is moving from focusing only on developer tools and infrastructure support to end-user applications such as Databases?*

Many researchers (e.g. [5, 28]) obtain data directly from repositories such as sourceforge. Despite having firsthand access to the data source, harvesting or crawling sourceforge could be a daunting task [12]. Alternatively, researchers may utilize subsidiary meta-data provided in a ***repository of repositories*** or **RoRs** such as FLOSSmole [11, 12]. FLOSSmole [18] may be described as RoRs or meta-repository of projects hosted at Freshmeat, sourceforge, `http://rubyforge.org/`, and `http://www.objectweb.org/`. From the FLOSSmole repository, we extracted *projects by topic* in five different topics- Database, Internet, Software Development, Communications, and Games/Entertainment- and developed time series analysis to study how projects in these topics grew from January 1st, 2005 to August 31st, 2006. We then compared the growth of open source database projects with projects in the other four topics. The rest of the paper is organized as follows. Section 2 presents trends in the use of RoRs in F/OSS research. In section 3 we present our research methodology and discuss our data collection and extraction. In Section 4 we present the results, discuss our findings, and list the validity threats to our research. Our conclusion and future work is presented in section 5.

2 Trends in F/OSS Research

Compared to traditional research practices under proprietary software, F/OSS development provides researchers with an unprecedented abundance of easily accessible data for research and analysis. A huge amount of data is available to study community participation in F/OSS projects [2, 10, 1, 9, 13] and developer and user involvement in projects mailing list [21, 25, 14, 15]. Web sites which host F/OSS projects also provide each project with repositories or tools to enable the collaborative software development process to proceed. The largest site which has generated a lot of research interest is sourceforge. Data from this site has been used to study many aspects of F/OSS. For instance, the geographical location of developers [20], topological analysis of developer communities [28], Knowledge collaboration across projects [17], patterns of software development [26, 8], percentage distribution of projects [22], etc. The traditional way of obtaining data for most of these kinds of research is spidering or crawling of sourceforge using Perl or Python scripts. However, instead of direct access to the sourceforge repository, researchers may also benefit from reusing data other researchers obtained from sourceforge. The University of Notre Dame maintains a data dump from sourceforge [5, 8] and other researchers may request and reuse the data in their studies [20]. In another study, [29] reused data from [12] to study the self-organizing patterns in wasp and F/OSS communities.

It is becoming increasingly evident that collecting and analyzing F/OSS data has become a problem of abundance and reliability in terms of storage, sharing, aggregation, and filtering [6]. Some of the problems researchers may face in obtaining and using data in their research can be summarized thus:

- **Convergence of data**: There is no standardized way of defining or a naming convention for variables in a repository. This may pose problems for researchers when it comes to harmonizing data across different repositories.
- **Without Notice!**: The data structure of a repository is held in a back-tier (database). And because many researchers just interact with the front-end of the repository, researchers can face a daunting task when a site or a repository maintainer changes the structure of the data or schema. [12].
- **Confidentiality:** Due to the sensitive nature of some aspects of the data (e.g. private emails), some projects might be reluctant to release some of their data at a time when the researcher actually needs it.
- **A friend of a friend (*FOAF*)**: A researcher not having direct access to the data he needs for his research may send a request to the project, either through mailing lists or to the repository maintainer. Experience shows that sometimes processing such a request is like waiting for rain in the desert. In this case, knowing someone who has obtained the data the research wants or knowing some members of the core team helps.

These difficulties significantly impede F/OSS research. As a result, many researchers see the need for the establishment and use of **R**epositories **o**f **R**epositorie**s** or RoRs. The whole concept of RoRs is an attempt to pull data from many and varied repositories and bring the data under one umbrella so that researchers can have easy access to data, reports, tools, and scripts used in F/OSS research. As [12] noted in their schematic analysis, current F/OSS research is "one-way traffic"; non-cyclical and non-collaborative. Once researchers obtain, analyze and publish their data, the product of their research is never put back to the community from which they obtained their data. Existing structures of software repositories do little to ameliorate this situation. The aim of RoRs should be to close this loop by encouraging researchers to contribute their data and any scripts and tools they used in their research to the RoRs from which they obtained the original data. Our view of how the RoRs concept should work is illustrated in Figure 1. From the diagram, note that there is a continuous feedback between the research community and the RoRs.

The first kind of RoRs available to researchers is FLOSSmole. For a detailed description of the purpose, design, and requirements of Flossmole, see [12]. Another RoRs in progress is the EU funded FLOSSMetrics (`http://www.flossmetrics.org/`) project. The FLOSSMetrics or Free/Libre Open Source Software Metrics project aims to construct, publish and analyze a large scale database with information and metrics about F/OSS development. Using existing methodologies and tools already developed the project will house data coming from several thousands of software projects. The project will also provide a public platform for validation and industrial exploitation of results. Some of the targets of the project are summarized:

- Identify, evaluate sources of data, and develop a database structure.
- Build and maintain an updated empirical database.
- Disseminate the results, including data, methods and software.

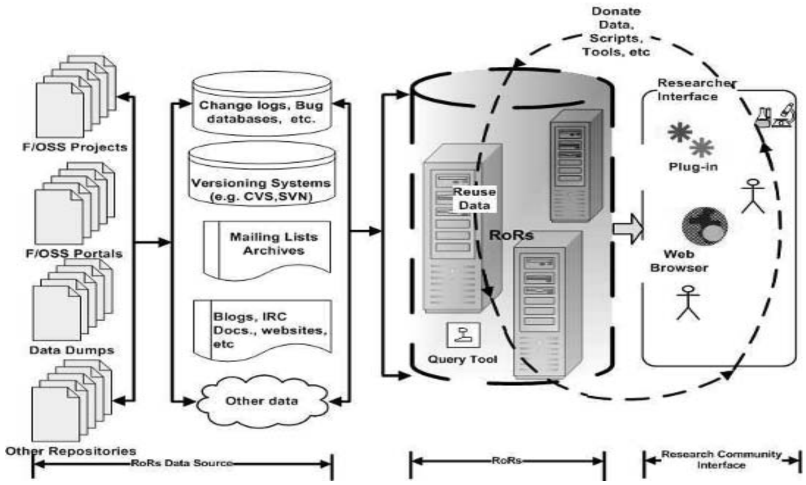


Fig. 1. Conceptual Framework of RoRs

The project will work with other projects such as FLOSSmole, the **S**oftware **Q**uality **O**bservatory for **O**pen **S**ource **S**oftware or SQO-OSS (`http://www.sqo-oss.eu/`) and **QU**ALity of **O**pen **S**ource **S**oftware or QUALOSS (`http://www.qualoss.org/`). Thus, it is becoming increasing feasible to use data from RoRs for quality F/OSS research.

3 Methodology

The research methodology employed in this paper is schematically shown in Figure 2. Where a similar methodology may be applicable is shown in dotted cones. The "Donate" arrows show a researcher contributing his Python script and the results of his analysis to the RoRs - FLOSSmole.

Data: The FLOSSmole repository has data dumps collected from repositories in text and excel files format. For our study, we downloaded raw and summary data of FLOSSmole's archives of sourceforge data during twenty months period, from January 1st, 2005 to August 31st, 2006. We chose five major *projects by topic*; Database, Internet, Software Development, Communications, and Games/Entertainment. We selected these five topics because, first, the last four show a dominant position in the sourceforge's "software Map" (Table 1) and will be quite representative of the types of projects being developed by the F/OSS community. Second, all the five topics or categories have been used in a previous study [22], and it will be interesting to compare and contrast our results with that study. Third, to validate our hypothesis whether there is evidence to support that F/OSS development is moving "up the stack" from developer tools and infrastructure support to end-user applications such as Databases, we will

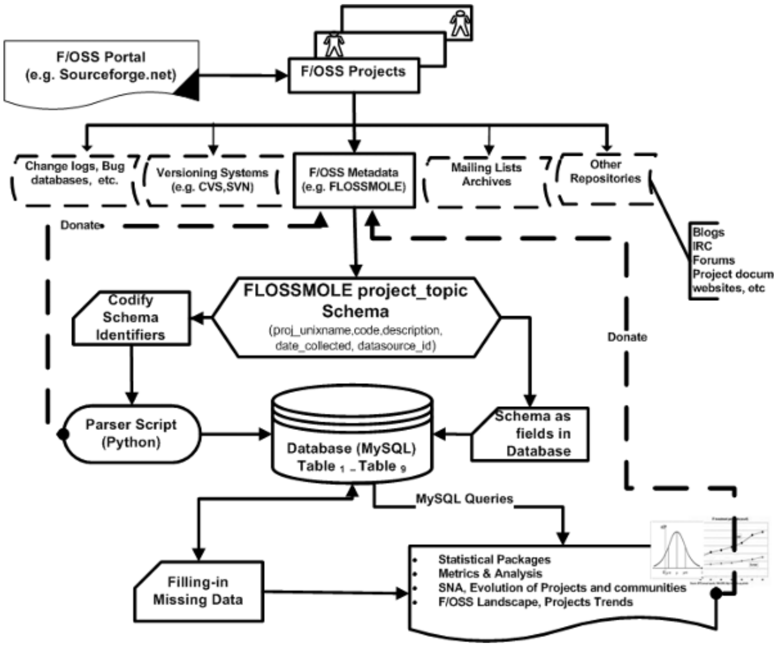

F/OSS Portal (e.g. Sourceforge.net)
F/OSS Projects
Change logs, Bug databases, etc.
Versioning Systems (e.g. CVS,SVN)
F/OSS Metadata (e.g. FLOSSMOLE)
Mailing Lists Archives
Other Repositories
Blogs
IRC
Forums
Project documents, websites, etc
Donate
FLOSSMOLE project_topic Schema
(proj_unixname,code,description, date_collected, datasource_id)
Codify Schema Identifiers
Donate
Parser Script (Python)
Database (MySQL) Table 1 - Table 9
Schema as fields in Database
MySQL Queries
Filling-in Missing Data
Statistical Packages
Metrics & Analysis
SNA, Evolution of Projects and communities
F/OSS Landscape, Projects Trends

Fig. 2. Methodological outline to extract data from FLOSSmole.

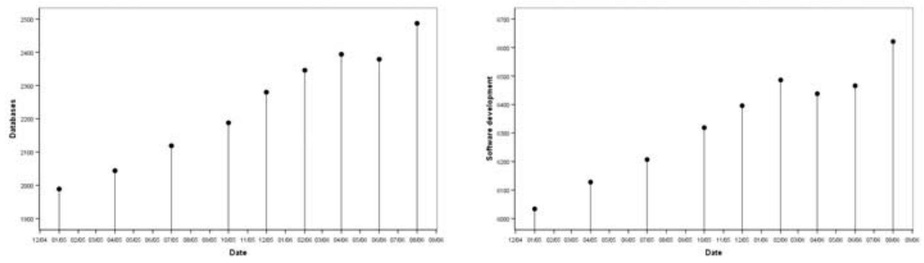
investigate how projects in the database topic scale against other projects in the other four topics.

Table 1. Number of projects in descending order in 9 out of 19 topics. Extracted from sourceforge's "software map" on 17/01/2007. Asterisks beside topics in our study.

Topic	Total Projects	Example
Internet*	26505	FileZilla
Software Development*	25840	Gaim
System	21524	phpMyAdmin
Communications*	17115	Gaim
Games/Entertainment*	15894	FreeCol
Multimedia	14426	MediaPortal
Scientific/Engineering	13542	K-3D
Office/Business	8802	Openbravo ERP
Database*	6509	phpMyAdmin
....	.....	.....

The FLOSSmole files we used include raw "Project Topic" data for each project. However, not every project in sourceforge lists this information [6]. We codified the "*project_topic*" schema obtained from the text files into a Python script and implemented it as fields in our MySql database containing nine tables, one for each available data. A text file of each month's data was parsed into the database for subsequent analysis. The database was queried for projects belonging to a given topic and the dates they were hosted at sourceforge.

Extracted data and Missing Values Estimation: Our original raw data had unequal time gaps. We had gaps of two months for the first year (2005) and gaps of one month for the second year (2006). For example, Figure 3 shows the dates and total numbers of projects extracted form the Database and Software Development topics.



(a) Database Topic (b) Software Devel. Topic

Fig. 3. Scatter plots showing gaps in the original data.

We exploited the linear trend in our data to fill the gaps with estimations. The "*linear trend at point*" method [23] was used to replace the missing data with the linear trend for that point. First, a linear regression line is fitted to the existing observations with respect to an index variable scaled from 1 to n (total number of observations) and then the missing values are replaced with their predicted values. The new data shown in Tables 2 have real or estimated values, in parentheses, for every month.

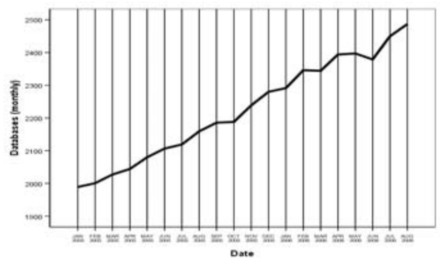
4 Results and Discussions

4.1 Overview of F/OSS Projects' Landscape

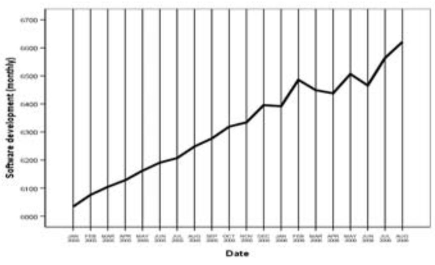
The representation of our data as time series with observations taken (or estimated) in equally spaced time intervals, gives us the opportunity to use time series analysis in order to construct predictive models or to study the behavior of our data in time. Time series plots for the Database and Software Development topics in figure 4 show that the number of projects in the F/OSS landscape is not at all stationary. Instead, projects in all the five topics exhibit a linear growth. Thus, we can use this behaviour to model and forecast the growth of the projects in all the topics.

Table 2. Number of F/OSS Projects by Topics.

Month	Database	Internet	Software Devel.	Communications	Games/Entert.
Jan. 05	1989	2465	6034	1990	2435
Feb. 05	(2001)	(2500)	(6076)	(2019)	(2444)
Mar. 05	(2027)	(2530)	(6104)	(2045)	(2479)
Apr. 05	2044	2546	6128	2064	2505
May 05	(2080)	(2590)	(6162)	(2097)	(2548)
Jun. 05	(2106)	(2621)	(6191)	(2123)	(2583)
Jul. 05	2119	2643	6207	2139	2602
Aug. 05	(2159)	(2681)	(6248)	(2176)	(2652)
Sep. 05	(2186)	(2711)	(6277)	(2202)	(2687)
Oct. 05	2188	2738	6319	2211	2700
Nov. 05	(2239)	(2772)	(6334)	(2254)	(2756)
Dec. 05	2280	2817	6396	2290	2778
Jan. 06	(2291)	(2832)	(6392)	(2306)	(2826)
Feb. 06	2346	2928	6486	2398	2887
Mar. 06	(2344)	(2893)	(6449)	(2359)	(2895)
Apr. 06	2394	2942	6438	2412	2961
May 06	(2397)	(2953)	(6507)	(2411)	(2965)
Jun. 06	2379	2916	6466	2377	2939
Jul. 06	(2450)	(3014)	(6564)	(2463)	(3034)
Aug. 06	2487	3043	6621	2483	3104



(a) Database Topic



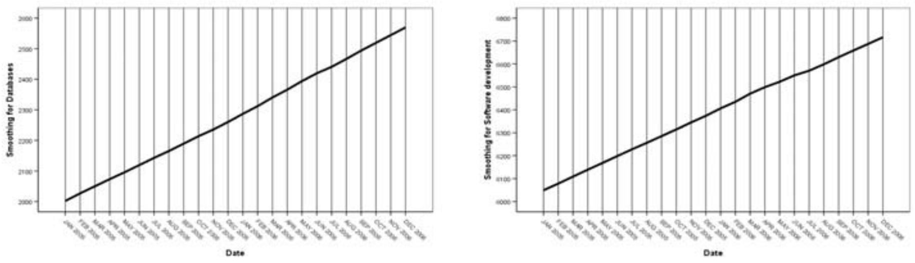
(b) Software Devel. Topic

Fig. 4. Linear growth of Projects by Topic.

Forecasting Growth: We applied exponential smoothers to model the behavior of our data and provide forecasts for the next three months, until the end of the year, 2006.future months. The smoothing curves in Figure 5 are almost straight lines due to the strong linear trend.

4.2 Are Database Projects Popular with the F/OSS Community?

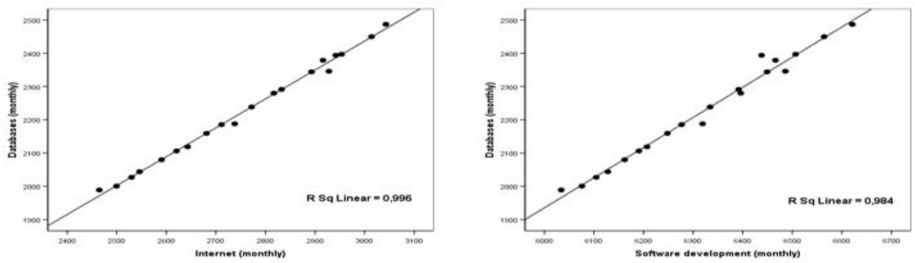
End-user applications such as databases are not a panacea or unknown quantities amongst the F/OSS community. Software developers, vendors, and database



(a) Database Topic (b) Software Devel. Topic

Fig. 5. Sequence plots showing smoothing curves with forecasts until the end of 2006.

users are already familiar with major players. MySQL, PostgreSQL, Berkeley DB, Firebird, and many others, continue to attract attention and their user base is growing. Projects in the database topic are doing well in the F/OSS landscape. Figure 6 shows that there is a significant linear correlation in the growth of Database projects and Internet (R^2=0.996) and Software Development projects (R^2=0.984). The growth trend is similar in the other topics.



(a) Database vs Internet (b) Database vs Software Development

Fig. 6. Trend in the growth of Database projects versus Internet and Software Development projects.

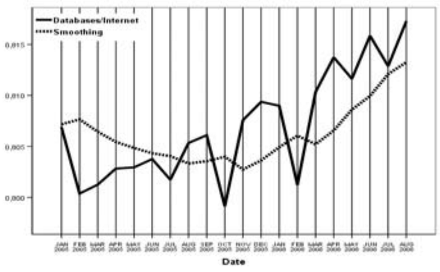
The Pearson correlation coefficients given in Table 3 shows that the linear correlation among all topics is highly significant as all are very close to 1. The corresponding hypothesis tests show that these correlations are significant at the 0.01 level.

In order to compare projects in the Database topic with projects in other topics, it is interesting to investigate the ratios obtained by dividing the number of database projects at each time period by the corresponding numbers of projects in other topics. Some of the sequence plots obtained by plotting these rations are shown in Figure 7.

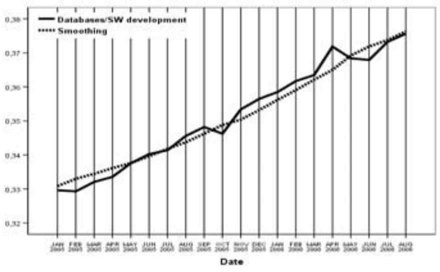
For a better understanding of the comparison, we added in each plot a smoothing curve showing more clearly the overall behavior. Plot (a) shows

Table 3. Correlation between projects in topics.

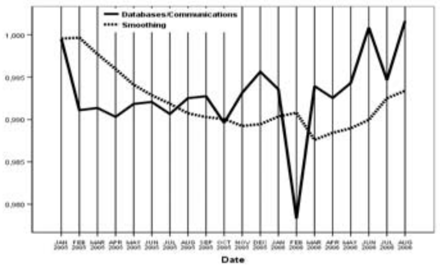
	Internet	Software devel.	Communications	Games_Entert.
Databases	.998	.992	.997	.999
Internet		.995	.999	.996
Software devel.			.993	.990
Communications				.995



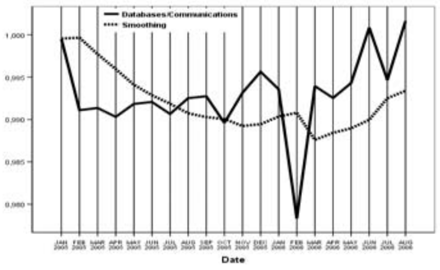
(a)



(b)



(c)



(d)

Fig. 7. Sequence plots showing ratios of database projects by each of the other topics.

that the number of database projects is approximately 81% of the internet projects but there seems to be a growing trend for this ratio from November 2005. Plot (b) shows that the database projects are around 35% of the software development projects but this ratio has a clear linear growth. Plot (c) shows that the number of database projects is almost the same as that of communications, since their ratio is very close to one. There is initially a descending trend but later the trend is ascending. Finally, plot (d) shows that database projects vary by, approximately, 81% of Games/Entertainment projects, but this ratio has an almost linear descending trend.

4.3 Hypothesis Validation

Hypothesis: *Is there evidence to support that F/OSS development is moving from focusing only on developer tools and infrastructure support to end-user applications such as Databases?*

It follows from the discussion presented in section 4.2 that F/OSS development is moving "up the stack" from developer tools and infrastructure support to end-user applications such as open source databases. The analysis shows a steady growth of not only end-user projects such as database but also growth in major areas such as Internet, Software Development, Communications, and Games and Entertainments. Thus, our hypothesis is supported within the limit of our analysis.

4.4 Validity Threats and Considerations

Any claim to map the ecology of the types of projects in the F/OSS landscape should be treated with caution for the reason that:

- *Ex-ante analysis* of our data shows that some projects belong to more than one topic (see Table 1). For example, phpMyAdmin is classified under the Database as well as Systems topic. Each topic also contains 'sub-topics', for example the Database topic also contained 'database Engines/Servers' and 'Front-Ends' as sub-topics. Thus, there is the inevitable consequence of counting some projects more than once, thus inflating the numbers.
- Even though sourceforge is the largest repository of F/OSS projects, there are other repositories (e.g. Freshmeat, Savannah.gnu) which are equally important.
- Not all F/OSS projects are hosted at sourceforge. In fact most of the 'successful' F/OSS projects are hosted outside sourceforge. Others only maintain a link with the portal.
- The quality of projects in sourceforge vary tremendously. A more plausible option would have been to define criteria for the types of projects to study. For instance projects by topic for projects with a certain number of programmers, downloads, sourceforge rating, etc)
- Small dataset. We based our discussion on a dataset obtained during 20 months. Perhaps 2-5 years data would have revealed a different and clearer trend than the one we reported.

The research methodology we employed in this paper may serve as an impetus for researchers faced with the inevitable consequence of missing data. We have also highlighted the importance and benefits of using RoRs in F/OSS empirical studies. However, important questions about RoRs need addressing:

Infrastructural/Technical: What are the requirements for implementing the basic infrastructure required to setup and link the repositories? What are the major problem associated with integrating the schemata of individual and/or heterogeneous databases [30]? What are the required communication protocols (OAI?)? How to deal with the issue of missing data? What are the lessons learnt from the technical challenges from the FLOSSMole?

Data Quality: How will obtaining data from many and different repositories (employing different schemas) affect the quality of the data? How to deal with issues of missing data?

Motivational/Social: Are researchers prepared to 'give back' their fine-tuned data, scripts and research tools to RoRs? How to create a partnership between RoRs and their parent repositories so that RoRs maintainers will be well informed should the structure of the parent repository change.

Economical: Many F/OSS projects are voluntary in nature and depend on benevolent donations from individuals to function. The establishment and maintenance of RoRs need financial funding. How will RoRs be funded? One initiative in this regard is the EU funded FLOSSMetrics project.

5 Conclusion

In this paper metadata from FLOSSmole, which is a repository of repositories (RoRs), was used to discuss the F/OSS landscape in terms of the projects being developed by the F/OSS community. We encountered gaps in our data and used the 'linear trend at point' method to fill the gaps with estimations. Various statistical methods were employed to investigate the F/OSS projects' landscape and we found out that projects in all the topics studied are growing linearly. Exponential smoothing curves produced almost straight lines due to the strong linear trend. Three months forecast showed that the number of projects in these topics continued to grow beyond our study. Comparing the trend in the growth of the number of projects in the database topic against the other four, revealed a high correlation between databases and all the other four projects' topics. These findings show that applications developed by the F/OSS community are not limited to infrastructural or Internet based components only, but also to end-user products such as Databases. The ration of database to Internet, software development, and Games/Entertainment projects showed a linear growth. Furthermore, projects in the database topic are growing almost at the same rate as those in the communications topic.

Future Work: We are spidering sourceforge to obtain data spanning many years so that we can build better prediction models to study the F/OSS landscape. We are also collecting data on researchers who are using the same dataset. We intend to develop social networks where researchers form nodes on a network/graph and two or more nodes are linked if they share the same dataset. Collaborative networks of this nature can reveal a great deal about the social structure of the F/OSS research community , such as the presence knowledge brokers [25].

Acknowledgement. This work is partially supported by the Free/Libre/Open Source Software Metrics and Benchmarking Study (FLOSSMetrics), Project No: 033982, (http://flossmetrics.org). Sincere thanks to the maintainers and participants of Flossmole. Script, data, and analysis used in this paper can be found at: `http://ossmole.sourceforge.net/`

References

1. Hahsler M, Koch S., Discussion of a Large-Scale Open Source Data Collection Methodology, Proceedings of the 38th Hawaii International Conference on System Sciences (IEEE, HICSS '05-Track 7), Jan 03-06, Big Island, Hawaii, 2005, page 197b.
2. Barahona, J. et al., Analysing the Anatomy of GNU/Linux Distributions: Methodology and Case Studies (Red Hat and Debian), In Koch, S. (Ed.), Free/Open Source Software Development, Idea Group Inc., 2005, pp: 27-58.
3. govtech.net, SourceForge.net Surpasses 100,000 Open Source Projects, `http://www.govtech.net/news/news.php?id=94043`, May, 2005.
4. Nikolai Bezroukov, Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism), Firstmonday, 1999, vol. 4(10).
5. Scott Christley and Greg Madey, Collection of Activity Data for SourceForge Projects, Technical Report: TR-2005-15, University of Notre Dame, 2005.
6. Megan S Conklin, Beyond Low-Hanging Fruit: Seeking the Next Generation in FLOSS Data Mining, In Damiani, E., Fitzerald, B., Scacchi, W., Scott, M., Succi, G (Eds.),IFIP International Federation for Information Processing, Open Source Systems, Boston: Springer, Vol. 203, 2006, pp: 261-266.
7. Kevin Crowston, Hala Annabi, and James Howison, Defining Open Source Software Project Success, Proc. of International Conference on Information Systems, ICIS 2003, 2003.
8. G. Madey, V. Freeh, and R. Tynan, The open source software development phenomenon: An analysis based on social network theory, In Americas conf. on Information Systems (AMCIS2002), 2002, pp: 18061813.
9. German, D. and Mockus, A., Automating the Measurement of Open Source Projects, ICSE '03 Workshop on Open Source Software Engineering, Portland, Oregon, May 3-10, 2003.
10. Ghosh, A.R., Clustering and dependencies in free/open source software development: Methodology and tools, Firstmonday, Vol. 8(4), 2004.
11. Howison, J., Conklin, M., and Crowston, K., FLOSSmole: A collaborative repository for FLOSS research data and analyses, International Journal of Information Technology and Web Engineering, Vol. 1(3), 2006, pp: 17-26.
12. J. Howison, and K. Crowston, The Perils and pitfalls of mining SourceForge, 26th International Conference on Software Engineering, Edinburgh, Scotland, 2004.
13. Koch S, Schneider G., Effort, cooperation and coordination in an open source software project: Gnome., Information Systems Journal, Vol. 12(1), 2002, pp: 27-42.
14. Krogh G, Spaeth S, and Lakhani, K., Community, joining, and specialisation in open source software innovation: a case study, Research Policy, Vol. 32, 2003, pp: 1217-1241.

15. Lakhani K, Hippel von E., How open source software works: "free" user-to-user assistance, Research Policy, Vol. 32, 2003, pp: 923-943.
16. David M. Nichols and Michael B. Twidale, Usability processes in open source projects, Software Process: Improvement and Practice, Vol. 11(2), 2006, pp: 149-162.
17. Masao Ohira, Naoki Ohsugi, Tetsuya Ohoka, and Ken-ichi Matsumoto, Accelerating cross-project knowledge collaboration using collaborative filtering and social networks, MSR '05: Proceedings of the 2005 international workshop on Mining software repositories, ACM Press, 2005, pp: 1-5.
18. FLOSSmole Project, FLOSSmole Project (2004-2006) Sourceforge, `http://ossmole.Sourceforge.net`.
19. Raymond, E.S., The Cathedral and the Bazaar. Musings on Linux and Open Source by an Accidental Revolutionary, OReilly, Sebastopol, USA., 1999.
20. Gregorio Robles and Jesus M. Gonzalez-Barahona, Geographic location of developers at SourceForge, In MSR '06: Proceedings of the 2006 international workshop on Mining software repositories, ACM Press, 2006, pp: 144–150.
21. Sulayman K. Sowe and Ioannis. Stamelos, Identification of Knowledge Brokers in F/OSS Projects through Social and Collaborative Networks, In Proc. of 10th Panhellenic Conference on Informatics, Volos, Greece, 2005, pp.285-303.
22. Sulayman K. Sowe, Ioannis Samoladas, and Ioannis Stamelos, Trends in Open Source Database Management Systems. In Laura C. Rivero, Jorge H. Doorn, and Viviana E. Ferraggine (Eds.) Encyclopedia of Database Technologies and Applications, Idea Group, Inc., 2005, pp: 457-462.
23. David J. Sheskin, Handbook of Parametric and Nonparametric Statistical Procedures, Chapman and Halucrc, 2004.
24. Sulayman K. Sowe, Ioannis Stamelos, and Karoulis, Anastesiou, A constructivist View on Knowledge Management in Open Source Virtual Communities. In Figueiredo, D. A, and Paula, A. (Eds.), Managing Learning in Virtual Settings: The Role of Context, Idea Group, Inc., 2005, pp: 290-308.
25. Sulayman K. Sowe, Ioannis Stamelos, Lefteris Angelis, Identifying knowledge brokers that yield software engineering knowledge in OSS projects, Information and Software Technology, 2006, vol. 48, pp: 1025-1033.
26. Katherine J. Stewart, David P. Darcy, and Sherae L. Daniel, Observations on patterns of development in open source software projects, 5-WOSSE: Proceedings of the fifth workshop on Open source software engineering, St. Louis, Missouri, ACM Press, 2005, pp: 1-5.
27. Walt Scacchi, Joseph Feller, Brian Fitzgerald, Scott A. Hissam, and Karim Lakhani, Understanding Free/Open Source Software Development Processes, Software Process: Improvement and Practice, 2006, vol. 11(2), pp: 95-105.
28. Xu J, Gao Y, Christley, S, Madey S., A topological Analysis of the Open Source Software Development Community, IEEE Proceedings of the 38th Hawaii International Conference on System Sciences, (IEEE, HICSS '05-Track 7), Jan 03-06, Big Island,Hawaii., 2005, page 198a.
29. Valverde, S., Theraulaz, G., Gautrais, J., Fourcassie, V., and Sole, R. V. Self-Organization Patterns in Wasp and Open Source Communities. *IEEE Intelligent Systems*, 2006; **21**(2): 36-40.
30. N. Bassiliades, I. Vlahavas, A. Elmagarmid, E. Houstis, Interbase-KB: A Knowledge-based Multidatabase Sustem for Data Warehousing, IEEE Transactions on Knowledge and Data Engineering, Vol. 15, No. 5, pp. 1188-1205, 2003.

Community Structure, Individual Participation and the Social Construction of Merit

Matthias Studer
University of Geneva, department of econometrics, 40 Bd du Pont d'Arve, 1211 Geneva 4, Switzerland, matthias.studer@metri.unige.ch

Abstract. FLOSS communities are often described as meritocracies. We consider merit as a social construction that structures the community as a whole by allocating prestige to its participants on the basis of what they do. It implies a hierarchy of the different activities (web maintenance, writing code, bug report...) within the project. We present a study based on the merging of two datasets. We analyze the archive of KDE mailing lists using a social network. We also use responses to a questionnaire of KDE participants. Results bring empirical evidences showing that this hierarchy structures the community of KDE by allocating more central position to participants with more prestigious activities. We also show that this hierarchy structures individuals participation by giving greater "membership esteem" to members involved in more prestigious activities.

Key words: Collective Self-esteem, Community of Practice, Meritocracy, Open Source, Social Network Analysis, Social Structure.

Introduction

It is often proposed that the distinctive social structure of FLOSS communities could be one of the key reasons of its success [1]. This organisation is often described as meritocratic [2] or at least, as having “an ideological commitment to meritocracy” [3]. In this article, we propose to discuss the concept of meritocracy and to describe how it structures the community. We intend to bring empirical evidences to support our matter by taking KDE[1] as a case study. Our study is based on two data sources: e-mails archives and a questionnaire of KDE participants.

Meritocracies are social systems in which the social position is determined by merit. In other words, the social position is determined by the social valorisation of the activities done in the community. Thus, some activities lead to more influential position than others according to the merit linked to each activity. This distribution of the merit between different activities structures the community as a whole. According to our point of view, the definition of the merit is not objective, but results from a social construction that we need to understand better. Indeed, it seems to form the base of power relationships in FLOSS communities.

According to the theory of communities of practice [4], this allocation of power is described using the concept of mutual engagement. What a person does allows to "categorize him as", i.e. to assign him to a peculiar social position. This classification (which can be either positive or negative) is carried out by the other members of the community in an unconscious way through the returns (comments) made on each contribution. But what we do also makes it possible to "categorize us ourselves as" through the returns carried out by the other members of the community. Thus, the process of "power allocation" is done at the community level by the allocation of power to people who deserve it. It is also an individual process since each one "categorize himself as" compared to other members of the community.

The socially constructed definition of merit structures community on two levels. Firstly, it structures the community as a whole by allocating more influential positions to members carrying out more valorised activities. Secondly, it structures individual participation by giving a conscience of oneself position in the community. Thus, the centre of our analysis is the "person-in-the-social-world" [5], i.e. we seek to clarify the relationships between socials norms and individual participation. According to the theory of communities of practice [4], we should look at the relationships between these two levels in order to better understand the social structure. The social structure acts through its internalization by contributors.

Following these two points of view, we selected two data sources. The first one is the archive of all e-mails archived by MARC since the beginning of KDE. This data source will enable us to approach the social structure as a whole through a social network analysis. The second data source comes from a questionnaire of KDE participants. It will enable us to approach the meaning given to each activity and some more subjective elements of our assumptions like membership esteem of contributors.

This article is organized in the following way. We present the data sources and the methods we used to extract relevant information from it. Then, we will analyze the hierarchical organization of activities. After that, we will look at the internalisation done by KDE contributors before concluding our article.

[1] See http://www.kde.org

Presentation of data sources

As we said, we have two data sources: e-mails archive and a questionnaire of KDE contributors. We used e-mails sent to KDE mailing-lists and archived by MARC[2]. We used the data from beginning of January 2006 until end of June 2006 covering a six month period. These e-mails come from the lists of discussions within each project and sub-project. We also used information from the "kde-bug-dists" mailing-list, a list gathering automatic notifications for all changes made to Bugzilla. The use of e-mails archive enables us to bring all these sources together on the basis of names and e-mail addresses. We used this information to build a social network analysis of all participants to KDE mailing-lists using Pajek [6]. In this kind of analyses, two definitions are essential: inclusion and relationship.

Regarding inclusion, two problems quickly arise: neither the e-mails addresses nor the names can be considered unique. Consequently, we used an in-depth search algorithm to put together the couples of "name-email". This algorithm was used to propose possible merges to a human. Since all regroupings were human-supervised, we were forced to use a selection criterion. We thus regrouped and selected all person having sent at least ten messages over a period exceeding one month. One can argue that we introduced a systematic bias in our analysis by using this selection criterion. However, we think that this bias will not be very influential. We are interested in relationships with people who are important for the community. It is not abusive to think that these peoples sent at least ten e-mails.

One usually defines the relation using "point-to-point" information such as the "In-Reply-To" header of emails. However, this information was not available in KDE archives. Thus, we used the definition of "thread" from MARC to constitute our network. We have then defined the relationship between two persons as:

> The relationship between a person A and B is equal to the sum of all messages sent by A in "threads" where B also sent at least one message.

The relationship has a direction (from A to B) and the value is different according to this direction. However, and this rises from the given definition, if A has a relationship with B, then B has a relationship with A. It will not be automatically the same value. The graph obtained is directed and valued. Our measurement also contains a scale about the "force" of the relationship. Thus, taking part in a discussion with a lot of different participants implies more "relations" than taking part in a small discussion. In other words, each message is not equal in our construction of the network. This corresponds to some logic. By taking part in a large discussion (which has more chance to be considered as important), one acquires a greater visibility than in a small discussion implying only two people.

Consequently, how to interpret the values of relationships? We suggest thinking in term of contacts. One "unit of relation" corresponds to one contact initiated if one thinks in terms of outgoing relationships, or with one contact received in the case of incoming relationship. The construction of our network makes difficult to compare values of incoming and outgoing relationships since the same message does not have

[2] Mailing list ARChives (http://lists.kde.org)

the same weight according to the direction: a message received corresponds to a contact whereas a message sent can correspond to several contacts. Being given that most messages are “functional”, we think that we should interpret our network as some kind of cooperation network. In our sense, it would be abusive to speak of friendly networks. In fact, the use of IRC discussions archive would be probably closer to such an interpretation of the network.

We computed several indicators from the social network analysis in order to test our hypothesis, namely the degree, the sum of incoming lines, the sum of outgoing lines and the maximum value of all arcs. The degree is simply the number of arcs connected to a given vertex (contributor) in our social network. According to our definition of the network, the sum of incoming lines corresponds to the number of messages received. The sum of outgoing lines can be interpreted as an indicator of influence in our network. Finally, the maximum values of all arcs (incoming or outgoing) should reflect the strength of the connection with other members.

All these indicators are local. So, we also computed “proximity prestige” [8]. This indicator is equal to the number of vertex that a given vertex can reach according to the arcs present in the network divided by the mean length of path to reach all these vertices. It is an indicator of the position in the global network.

We also used a questionnaire of KDE participants. The answers were collected online. Thus, the answerers were voluntary. KDE contributors were informed about the existence of the questionnaire through several messages sent on KDE mailing-lists. We took care to include all kind of mailing-lists such as users-oriented, translators and developers mailing-lists. We also took care to address our questionnaire to all kind of KDE contributors (including bug-reporter, translators, developers, etc.). However, we did not send the advertisement to all mailing-lists – we did not want to be considered as a “spammer” by KDE contributors.

We collected 131 answers. This low number of answers leads us to think that generalization of our results should be done with caution. Because we are here within a framework of observation and not of experimentation, the variations could be caused by factors for which we do not control. Answering the questionnaire was not especially long (approximately 15 minutes). However, 25 peoples did not answer the whole of the ten pages presented. The first question made it possible to establish the link between the questionnaire and the e-mails archives. This question was optional since some answerers may prefer to remain anonymous on Internet. Only 95 people gave an answer to this question.

The questionnaire was made up of questions about involvement in KDE, collective identity, demographic information and individual motivations. In this article, we will use the answers to three questions which were related to:

- The activities carried out within the community on a 6 item scale of frequency ranging from “Never or almost never” to “Every day or almost every day”.
- The prestige granted to each activity on a scale ranging from 1 (No prestige) to 10 (very prestigious).
- Two questions related to the importance that one think one have in a given social group. These questions were “I am a worthy member of my KDE group(s)” (positive formulation) and “I feel I don't have much to offer to my KDE group(s)” (negative formulation) on 7 item scale ranging from “Strongly disagree” to “Strongly agree”. These questions were taken from the “membership

esteem" subscale of the collective self-esteem scale proposed by Luhtanen and Crocker [7]. We transformed the formulation of the scale by replacing references to "social groups" with "KDE groups". The "membership esteem" subscale is equal to 8 plus answer to the first question minus answer to the second one.

We can use the data from the social network analysis to get an idea of the representativeness of our answerers. Unsurprisingly, our population is not representative but far more involved in the community. The mean degree of all network members is 10.49 against 54.64 for our answerers. This difference is statically significant and independent of the network indicator used. To get a better idea of the involvement of our answerers, we present on the figure below the maximum frequency between all activities carried out in KDE. As we can see, almost 75% of our respondents contribute at least "once or twice a week". Hence, most of our respondents can be considered as frequent contributors.

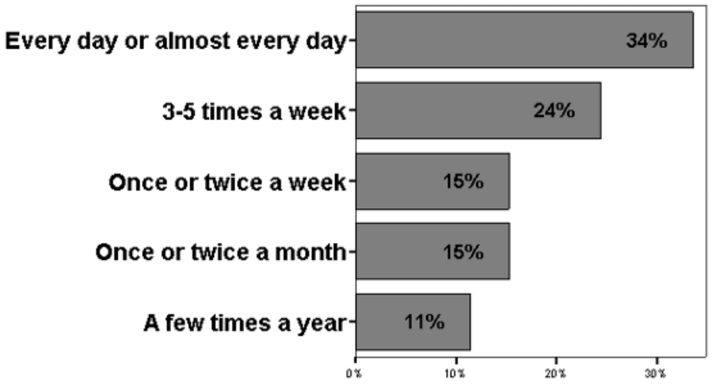


Fig. 1. Maximum frequency of involvement of answerers

In this article, we are interested in the social structure of the community. According to our point of view, the social structure comprises frequent as well as infrequent contributors. Thus, in following analysis, we included all answerers[3].

After having presented our data source, we will look at the social valorisation of activities and how these valorisation' schemes structure the community.

Hierarchical classification of activities

We argued that the social construction of merit organize the activities into a hierarchy. We propose to describe this organization before showing how it organizes the community as a whole. After that, we'll show how this social construction structures the individual participation of contributors.

[3] However, interpretations (and main results) do not change if we include only respondents contributing at least "once or twice a week".

The table below presents the mean prestige accorded to each activity by the answerers on a one (no prestige) to ten (very prestigious) scale. Seven answerers refused to answer this question by giving ten to all activities. The classification does not fundamentally change if we use a transformation such as rank. Changes in the hierarchy may appear if prestige averages are close[4]. The third column shows a rank for each activity. A difference of rank indicates a statically significant difference in the prestige distributions[5]. On the contrary, if the ranks are the same, we can not assume that one of the distributions is significantly higher.

Table 1. Hierarchical classification of the activities according to prestige scale

Activity	Average Prestige	Rank
Code	8.46	1
Coordination	7.44	2
Discussion about future development	6.98	3
Art	6.11	4
Bug Management	5.43	5
Help	5.42	5
Documentation	5.19	5
Translation	5.10	5
Packaging	5.03	5
Web	4.55	6
Bug Reports	4.29	6

According to our hypothesis, this ranking of activities is a social construction. It does not mean that some activities are less valuable or less important than others, but that the social interpretation and meaning given to these activities are different. This construction is not arbitrary but corresponds to some logic that we seek to understand.

The activities turned towards technology seem to be the most prestigious. Thus, the writing of code occupies the first position and the discussion of future developments the third. The presence of the activity "Art" in fourth position shows us that the creative and productive activities are largely valorised. Finally, let us note that the coordination of the community is seen as a prestigious activity. This is not surprising since our societies (as a general rule) tend to valorise this kind of activities.

The activities which come after are more difficult to distinguish, because the averages are close and the ranks are the same. It shows us that we can not assume a clear hierarchy between these activities. In this group, we find other form of contribution to the community such as bug management, user assistance, documentation, translation and packaging. It will be noticed that the user assistance appears relatively valorised for an activity which is not productive (or whose result is

[4] We always check all possible transformation in the reasoning presented below.

[5] We computed the Wilcoxon signed ranks test for all pair of activity scales. We then set the activity ranks according to statically significant (at the 5% level) difference in scale's distribution.

not visible in the final product). It seems to occupy a similar position to much more productive activities such as translation or documentation.

Finally, we find a last group with much less valuated activities with an average below five which would correspond to the mathematical average: website maintenance and bug reports. In our sense, the rank of website maintenance is quite surprising since it's a productive activity quite important for the community. Finally bug reporting occupies the lowest position in the hierarchy.

Translation is not classified in a uniform way between the answerers. Translators (N=54) tend to classify this activity significantly higher (Fisher's test: F=10.59; df=1; p=0.002) by giving an average of prestige of 5.85 whereas the "non-translators" (N=55) gives an average of 4.38. The translation passes thus from the fourth place (for translators) to the bottom of the hierarchy (for other contributors). This difference is even more significant if one takes into account the people making translation at least "one or twice a week" (N=32). The average of prestige is then 6.5 for translators against 4.52 for the others. The differences in distributions (Wilcoxon test) are statistically significant (Z=-3.291; p=0.001) in the first case and the second (Z=-3.841; p<0.0001).

This difference is important. It means that non-translators will evaluate less prestigious "translation" than translators. Thus, translators will gain much more prestige inside their own sub-community than in the community as whole. It shows us that translators seem to form a sub-community with their own definition of merit. However, this definition is not totally different. The top of the hierarchy is not disputed. Translation does not imply what translators consider as the most prestigious activities such as coding. Thus, translators are in a dynamic where their own activity (and their sub-communities) remains necessarily peripheral. We can also give a second interpretation of this difference. There is a social necessity for translators to valorise their own activities in order to maintain a regular involvement. Indeed, translation seems to be less valuated by the community as whole.

The activities performed inside KDE are not evaluated in a uniform way. Some activities are more prestigious than others. Therefore, there is a social construction of merit. In this social construction, technical and creative activities seem to be the most prestigious alongside with coordination. After having presented the valorisation's scheme of the activities, we will show how these differences of prestige structure the community as whole.

Activity Prestige and Community structure

According to our hypothesis, the social construction of merit should structure the community as a whole. So, we computed the correlation between the frequency of each activity and indicators computed from the network analysis. The correlations presented in the table below are Kendall's τ-b correlations[6] and are all significant at 1% level. In the table below, non-significant correlations are ignored.

[6] We always computed the Kendall's τ-b correlation rather than Pearson or Spearman correlation. Kendall's τ-b correlation is known to better handle ties values (which are typically frequent with ordinal measure) and make no assumption on the distribution of variable or on the form of the relationship between both variable (such as linear

Table 2. Correlation (Kendall's τ-b) between network indices and frequencies of activities

	Degree	Sum of incoming lines	Sum of outgoing lines	Maximum values of all arcs	Proximity prestige
Help	0.228	0.233	0.217	0.223	0.207
Code	0.400	0.421	0.416	0.428	0.377
Discussion	0.436	0.473	0.456	0.484	0.422
Translation					
Doc.	0.238	0.237	0.224	0.227	0.231
Art					
Web	0.337	0.332	0.324	0.304	0.318
Coordination	0.255	0.275	0.262	0.285	0.245
Bug Management	0.404	0.414	0.414	0.439	0.390
Bug reports	0.244	0.220	0.249	0.226	0.261
Packaging					

We expect all correlations to be significant and positive. Whatever we do, if we do it more frequently, we should have more relationships in the network and our relationships should be stronger[7]. By looking at the table below, we can distinguish three groups of activities:

- Code, discussion and bug management show strong correlations. We should notice that the first two correspond to the most prestigious activities.
- Coordination, Help, Documentation, web and bug reports show correlations around 0.25. This set of activities is in the middle of our hierarchy. The only exception is coordination but we had only few answerers who stated doing it.
- Art and Packaging activities show no significant correlation. But this is mostly because of the small number of answerers who stated doing it. This is not the case of translation which does not show any significant correlation.

The correlations seem, generally speaking, to follow the prestige accorded to each activity. However, "bug management" and "web" show much stronger correlations than expected. One possible explanation is that these activities come alongside with influential position. In other words, it is possible that influential positions imply responsibilities and activities that are not necessarily prestigious or "fun". In these cases, the influential positions would not be the result of such activities, but from the others performed alongside. Hence, "bug management" is highly correlated to "code" (τ-b =0.47) and "web" is mostly performed by long time contributors.

relationship for instance). This coefficient is known to be more conservative than the other and values are typically lower than for Pearson correlation. See Arndt et al. [9] for a full discussion.

[7] Some network indicators show stronger correlations with the frequency of each activity than others. The "sum of incoming lines" and the "maximum values of all arcs" show the strongest correlations. These indicators take into account the values of the arcs. Hence, it's not only the number of relations (degree) but also the values of the arcs that are important. Correlation between activities and proximity prestige are weaker than with other indicators.

The absence of correlation between network indicators and “translation” is interesting. Translation had a peculiar position in the hierarchy of the activities: non-translators were evaluating this activity as less prestigious than translators. This absence of correlation means that performing more often translation does not lead to more connections in the community. In other words, translation seems to be a peripheral activity. One can argue that social power is poorly linked with social network indicators. In our sense, we should interpret these indicators as a necessary condition. It is necessary to have different and strong link in order to exercise some sort of social power.

The socially constructed definition of merit seems to structure the community by allocating more central position to people performing more prestigious activities. According to our hypothesis, this construction of merit should also be internalized by contributors. This is what we will try to show now.

Activity Prestige and Individual Participation

For recall, membership esteem refers to the importance that one thinks one has within social groups to which one is identified. Membership esteem is strongly correlated with the indicators from the network (τ-b = 0.38 with the sum of incoming line; $p<0.0001$). Hence, members more strongly involved have more chance to feel important for the community. It is not necessarily the number of relations which counts more, but also the force of these bonds. Thus, one observes a positive correlation with the value of the strongest relation (τ-b=0.40; $p<0.0001$).

Table 3. Correlation (Kendall's t-b) between activities and membership esteem

	Membership esteem	Positive formulation	Negative formulation
Help	0.18*		-0.19 *
Code	0.38 **	0.27 **	-0.38 **
Discussion	0.39 **	0.35 **	-0.31 **
Translation			
Doc.	0.21 *		-0.19 *
Art			
Web	0.19 *		-0.23 **
Coordination	0.22 **		-0.26 **
Bug Management	0.23 **	0.17 *	-0.23 **
Bug reports			
Packaging			

** Significant at 1% level; * significant at 5% level.

Membership esteem is not only linked to network indicators but also with the activities done by contributors. Indeed, if we look at correlations on the table above, one can identify three groups of activities:

- The activities “codes” and “discussions” show the most important correlations with a value around 0.4.
- The other activities come then with correlations of about 0.2. These correlations are mainly the fact of the rejection of the negative formulation rather than of strongest acceptance of the positive formulation. Thus, we should conclude that

this is the expected correlation between frequencies and membership esteem. The only exception is “coordination”. Indeed, we had only few answerers that state doing it.

- Finally, “translation” and “bug report” do not show any significant correlations; the same applies to “packaging” and “art”. But these last ones do not have enough answerers to enable us to deduce something from it.

The first group of correlation shows us that the most prestigious activity comes together with stronger membership esteem. The second group of activities shows the expected relationships between activity and membership esteem. The presence of the last group is interesting. According to our assumptions, we should expect positive correlations. More one makes, in a given activity, better the membership esteem should be. This absence of correlation indicates us that this hypothesis is not verified regarding “translation” and “bug report”. In other words, doing these activities more frequently does not lead to greater membership esteem. These activities were also less valuated in the global classification. Therefore, we can conclude that membership esteem is linked to valorisation’s scheme of the activity.

We should clarify some points. Our results do not show that “translation” or other activities are devaluated within FLOSS communities. In fact, membership esteem was usually high and we should remember that the frequency of interaction (maximum frequency found between all activities) is highly correlated with membership esteem (τ-b=0.39). Therefore, all activities lead to greater membership esteem. Hence, our conclusion is that doing less prestigious activities contributes less to membership esteem than very prestigious activities.

We showed that the social construction of merit structures individual participation. Doing more prestigious activities contributes more to membership esteem than doing less valuated activities. The social construction of merit does not only structure the community as a whole. It is also internalised by contributors.

Conclusion

By using two distinctive sources of information, namely a social network analysis and a questionnaire of KDE contributors, we brought a new insight of FLOSS communities’ structuration since we were able to locate answerers inside the social structure. These two sources allowed us to think on two levels: the individual participation and the community. It also enabled us to think the relationship between these two levels of analysis which are usually measured separately.

We showed that there is a social construction of merit that implies a hierarchy of the activities performed in the community. This construction valorises the activities turned toward technological development such as coding and discussion about future development. Creative and coordination activities are also valorised. This does not mean, by any way, that some activities are less valuable or less time-consuming. We presented the social interpretation of activities not an evaluation.

We showed that the whole community is structured according to the activities performed and their social valorisations. The most prestigious activities seem to lead to more central position in the social network whereas we did not find such link for less prestigious activities. We noticed that some less prestigious activities (bug management and web maintenance) seem to come alongside influential position. We showed that different activities relate to different social positions as measured with social network indicators. Therefore, we can conclude that the social power also comes from the activity performed and not only from the frequency of interaction with the community. In other words, aside from frequency, the kind of activity performed is also a key dimension of social position.

This structuration dynamics is not only observable at the community level. We showed that it seems to be internalized by contributors. Esteem of its own importance within the community is linked with the kind of activity performed. Specifically, some activities seem to be more linked with membership esteem than others. The classification of the activities is internalized and not only an external factor of individual participation.

We showed that the social construction of merit structures the community as whole as well as individual participation of contributors. The KDE community can be described as meritocratic. However, we did not explain the process in detail. A more in depth or ethnological analysis is needed in order to precisely describe the social construction of merit. More precisely, we need to understand how the different contributions (within a given activity) are evaluated. Our analysis showed us that the social construction of merit defines which activities are linked with more influential social positions.

Beyond the structuration of KDE according to the social definition of merit, our analysis showed us that individual participation to a FLOSS project should be understood in relation with the social structure of the community. Contributors internalise the social structure of the community and the social structure influences their own participation. From a theoretical and methodological perspective, we should think the relationships between individual participation and social structure. Activities done inside a FLOSS community are not individualistic but a form of participation.

References

1. K. Crowston and J. Howison, The social structure of free and open source software development, *First Monday* **10**(2), (2005).
2. E. S. Raymond, Homesteading the noosphere, *First Monday* **3**(10), (1998).
3. J. Howison, K. Inoue and K. Crowston, Social dynamics of free and open source team communications, *in proceedings* The Second International Conference on Open Source Systems (2006).
4. E. Wenger, Communities of Practices: Learning, Meaning, and Identity (Cambridge University Press, Cambridge, 1998).
5. J. Lave and E. Wenger, Situated learning: legitimate peripheral participation (Cambridge University Press, Cambridge, 1991).
6. V. Batagelj and A. Mrvar, Pajek 1.14 – Program for Large Network Analysis' http://vlado.fmf.uni-lj.si/pub/networks/pajek/
7. R. Luhtanen and J. Crocker, A Collective Self-Esteem Scale: Self-Evaluation of One's Social Identity, *Personality and Social Psychology Bulletin* **18**(3), 302-318 (1992).

8. W. de Nooy, V. Batagelj and A. Mrvar, Exploratory Social Network Analysis with Pajek (Cambridge University Press, Cambridge, 2005).
9. S. Arndt, C. Turvey and N. C. Andreasen, Correlating and predicting psychiatric symptom ratings: Spearmans r versus Kendalls tau correlation, *Journal of psychiatric research* **33**(2), 97-104 (1999).

OpenBQR: a framework for the assessment of OSS

Davide Taibi[1], Luigi Lavazza[12], and Sandro Morasca[1]
1 Università dell'Insubria
luigi.lavazza@uninsubria.it, sandro.morasca@uninsubria.it,
davide.taibi@uninsubria.it, WWW home page: http://www.uninsubria.it
2 CEFRIEL
WWW home page: http://www.cefriel.it

Abstract. People and organizations that are considering the adoption of OSS, or that need to choose among different OS products face the problem of evaluating OSS in a systematic, sound and complete way. While several proposals concerning the evaluation of costs and benefits exist, little attention has been given to the evaluation of technical qualities and, in general, to the "usage-oriented" issues. In this paper the existing proposals are examined, the different types of qualities and issues that are relevant to potential users are described, and a coherent and innovative method for the evaluation of OSS is proposed. The proposed method is expected to support the potential user in the evaluation and choice of OSS in a flexible way, taking into account all the aspects that are relevant to the user.

1 Introduction

Open Source Software is a continuously growing movement. In order to give an idea of the size of the phenomenon, note that at the end of 2006 there were over 100,000 ongoing OSS project based on the best known repositories (such as SourceForge, CodeHaus, Tigris, Java.net and Open Symphony). OSS can also boast of several success stories: programs like the Apache projects, Netscape/Firefox, Eclipse, Linux, MySQL, and several others are well known and used by a huge number of people worldwide. Nevertheless, there are several areas where OSS was not adopted, at least not as widely as it could be expected. An example is given by the so called desktop environments and office applications. In fact, even in the areas where OSS has been successful, there are several potential users that did not adopt OSS.

Understanding why the adoption of OSS is limited is quite complex. A first reason is that the very concept of Open Source is hardly understood [1] [2] . People tend to confuse OSS with free software (i.e., software that can be used without paying any fee) and open standards with proprietary disclosed software (like PDF) [1]. Another reason is that it is not obvious how to carry out the cost/benefit analysis, given that the acquisition cost of OSS is usually null. Recently, the concept of Total Cost of Ownership (TCO) has been proposed as a mean to evaluate the cost of adapting, managing and maintaining OSS; nevertheless, the concept of TCO is not widely used, partly because it is not well understood (there are several, often not coherent, definitions) and partly because there is the suspect that most published TCO evaluations are driven by software vendors who want to convince customers that the commercial option is economically profitable. Finally, deciding the adoption of OSS requires the evaluation of the qualities of candidate OS programs, and their comparison with commercial programs. However, assessing the qualities of OSS is still a practice not well consolidated. Organizations facing the problem of deciding about the adoption of OSS have hardly any guide for carrying out a well structured comprehensive evaluation.

On the other hand, the producers of Open Source software cannot rely on clear indication concerning the factors that could determine the success of their products.

In this paper we discuss the qualities of OSS that determine its success and the features of OSS that should evaluated by potential users and adopters. Based on these considerations, a framework for the assessment of OSS is proposed. The goal is that such framework explicitly describes the qualities and properties of OSS that are considered important by both users and producers. In this way the framework can be employed by potential users for evaluating OSS. On the contrary, producers will get indications of what users value more, thus understanding what needs to be improved in their proposals.

The paper is structured as follows: Section 2 presents the current situation and the most recent proposals concerning OSS evaluation. Section 3 describes the features of OSS that –according to our analysis and understanding– are deemed important by organizations and professional users. Based on these considerations, our proposal for an OSS evaluation framework –named OpenBQR– is described in Section 4. In Section 5 we describe the validation activities that we carried out in order to confirm the capability of OpenBQR to represent the important features of OSS. Section 6 describes a web-based tool for carrying out the evaluations according to the criteria defined by the OpenBQR. Finally, Section 7 draws some conclusions.

2 State of the art and related work

The economic perspective

The first and most obvious problem with OSS is to assess its cost. Often OSS is free, i.e., there is no fee to pay in order to use the software; however, even in these cases it is clear that using OSS requires some investment. TCO (Total Cost of Ownership) addresses the evaluation of the cost of adopting and using a software program, including all the expenses, and spanning the whole lifecycle of the system [8]. Therefore, TCO involves the evaluation costs due to acquisition, adaptation, deployment, training, operation, maintenance, etc.

TCO applies to both OS and commercial software, thus allowing the comparison of costs. In fact, TCO became popular also because it was used to support both the thesis that OSS is more convenient than commercial software, and the vice versa.

Although TCO had the merit of providing a sound and comprehensive basis for the evaluation of SW costs, it is limited with respect to two important issues:

- TCO does not address the costs that are connected with the evolution of the user's business process, which could require updating the software or even changing it, thus calling for additional investments.
- TCO does not include the evaluation of benefits, thus providing an incomplete view of the financial consequences of adopting the considered software.

Other proposals have addressed these limitations of TCO. In order to take into consideration the future evolution of the users' needs, Cosenza proposed the Total Account Ownership (TAO) index, which aims at representing the degree of freedom of the user with respect to the technology provider [7]. The TAO considers issues like contracts and licenses, software adaptability, openness of formats and interfaces, documentation, training and assistance providers, etc., and indicates to what extent adopting a given piece of software is a commitment for the future.

The Full Business Value (FBV) aims at representing the whole value of the investment and includes the assessment of: system efficiency; system effectiveness; business efficiency; business effectiveness. The TCO can therefore be seen as a means to prove part of the information required by the FBV.

However, none of the TCO, TAO and FBV indexes address the issue of software quality. Since the adequacy of the software –from both the functional and quality point of view– is of fundamental importance, it is clearly necessary to assess them.

Next section discusses the evaluation of technical qualities as well as the assessment of the software adequacy with respect to the business process it is supposed to support.

The quality perspective

Recently, the problem of evaluating OSS became evident, so that a few organizations invested some effort in the creation of models for the quality and evaluation of OSS. The variety of models proposed witnesses the attention for the problem, but also demonstrates the difficulty of defining a fully satisfactory model.

The Open Source Maturity Model (OSMM) [3] is an open standard that aims at facilitating the evaluation and adoption of OSS. The evaluation is based on the assumption that the overall quality of the software is proportional to its maturity.

The evaluation is performed in three steps:

1. Evaluation of the maturity of each aspect. The considered aspects are: the software product, the documentation, the support and training provided, the integration, the availability of professional services.
2. Every aspect is weighted for importance. The default is: 4 for software, 2 for the documentation, 1 for the other factors.

3. The overall maturity index is computed as the weighted sum of the aspects' maturity.

The OSMM has the advantage of being quite simple. It allows fast (subjective) evaluations. However, the simplicity of the approach is also a limit: several potentially interesting characteristics of the products are not considered. For instance, one could be interested in the availability of professional services and training, in details of the license, etc. All these factors have to be 'squeezed' into the five aspects defined in the model.

In general we doubt that using 'maturity' as a proxy of the overall OSS quality is a good idea. Since we are interested in the evaluation of the OSS quality, it is much more effective to go straight for the definition of metrics that represent directly the aspects of the SW product that determine the quality for the user, i.e., what the users consider important in order to make OSS suitable for usage.

The Open Business Readiness Rating (OpenBRR) [5] is an OSS evaluation method aiming at providing software professionals with an index applicable to all the current OSS development initiatives, reflecting the points of view of large organizations, SMEs, universities, private users, etc. On the official Open BRR site several evaluations are available. They can be examined and easily adapted: you just need to input the parameters that suit best your needs in the spreadsheet containing the evaluation. The proponents of the method plan to apply it to all SourceForge and Java.net projects, so that potential users can find a ready to use evaluation of the software they are interested into.

In the first step of the evaluation, the list of programs to be evaluated is compiled. Then every component is evaluated with respect to a set of indicator selected according to the target usage and including: the type of license, the compliance with standards, the existence of a user base, the availability of reliable support, the implementation language, internationalization, etc. Then the functionality of products is evaluated. The features of a "reference application" are identified and their importance is graded with respect to "standard usage". Then every product is evaluated with respect to how well it implements every feature. Finally, the grades are normalized and the final evaluation (a grade in the 1..5 range) is computed.

The Open BRR is a relevant step forward with respect to the OSMM, since it includes more indicators, the idea of the target usage, and the possibility to customize evaluations performed by other, just by providing personalized weights. With respect to the latter characteristics, the Open BRR as however some limits: one is that for many products it is difficult to choose a "reference application" that reflects the needs of all the users; another is that there are lots of possible target usages, each with its own requirements; finally, every subjective evaluation performed by a user could be not applicable to other users. In any case, the final score is probably a too synthetic indicator to represent the complex set of qualities of a software product.

Qualification and Selection of Open Source Software (QSOS) is a model for the selection and comparison of OS and free software [4]. The evaluation process is carried out in four independent iterative phases. The definition phase aims at identifying the factors to be considered in the following phases. Phase 2 aims at collecting from the OS community the relevant information concerning the products. The goal is to create for every product an identity card (IC) reporting general information (name of the product, release date, type of application, description, type of license, project URL, compatible OS, …), available services, functional and technical specifications, … The quality aspects of the selected products are evaluated, and a grade (in the 0..2 range) is assigned according to the evaluation guidelines provided by QSOS. Phase 3 is dedicated to the definition of the selection criteria. The user's needs and constraints are described. Phase 4 consists in the comparison of the products' evaluation forms with the selection criteria, and in the identification of the product that matches betters with the user's needs and constraints.

Although in principle the method is effectively applicable to most OSS, the QSOS approach does not represent a relevant step forward with respect to other evaluation methods. Its main contribution is probably the explicitation of the set of characteristics that compose the IC, and the provision of a guideline for the consistent evaluation of these characteristics. Nevertheless, the evaluation procedure is too rigid and a bit cumbersome. For instance, it is required to define the IC of products that could be filtered away in phase 4 because they do not match the requirements. Such a procedure is justified when the ICs of products are available from the OS community before a user begins the evaluation. However even in this case it may happen that the user needs to consider aspects not included in the IC: this greatly decreases the utility of ready-to-use ICs. The strict guidelines for the evaluation of the IC, necessary to make other users' scoring reusable, can be ill suited for a specific product or user. Finally, even though in the selection criteria it is possible to classify requirements as needed or optional, there is no proper weighting of features with respect to the intended usage of the software.

3 Features of OSS that determine its acceptance by professional users

Assessing Open Source Software can require a complex process. In this Section, we describe the characteristics that are taken into account by people in order to assess the overall quality of OSS when choosing and adopting an OSS.

After a complete analysis of requirements, a set of parameters should be assessed, which favour a complete comprehension of the OSS being evaluated. We have identified several straightforward indicators that clearly show the quality of a software package to be adopted, divided into five different areas: functional requirement analysis, target usage assessment, internal quality, external quality, and likelihood of support in the future.

Target usage assessment

License: Not all open licenses are equal. Some licenses are more restrictive than others. If you need to extend the software, copy left properties are important because they allow modification of the code base and the redistribution of the modified version as long as the new product stays open.
Compliance with standards: for several application domains compliance with standards is important. For instance, in a website implementation, valid W3C-HTML code is a first step toward more compatibility with browsers and better rendering of pages. Using only strict HTML (that is, the Strict HTML DTD) makes the site easier to maintain and evolve.
Implementation language: if customization work and internal support are required, it is important to choose a product written in a programming language that is sufficiently mastered by the organization's programmers.
Internationalization Support: useful for applications that need to be translated into different languages.
Books: the availability of books about the software is a strong indicator of the software's level of maturity and popularity.
Interest by well known industry and market analysts and consultants: the availability of research reports on the software by analysts from leading market research firms (like Gartner or IDC) usually witnesses the relevance and diffusion of products.

Internal quality

With OSS it is possible to examine the internal quality of software, which is generally not disclosed for commercial software. For the purpose of evaluating the internal qualities you can choose among the many metrics proposed in literature and effectively supported by tools, like McCabe Cyclomatic Complexity, Chidamber and Kemerer's object-oriented metrics suite, Halstead complexity metrics, etc.

External quality

The main indication for the external quality of a software product is the defect density. It is therefore interesting to evaluate the number and severity of bugs over time, as well as defect removal speed. The latter is also a good indicator of the quality of support for the product.

In some cases it is also relevant to evaluate the defect removal process. In some cases, the removal of specific defect can be sped up by "donations" (in practice, you pay the organization maintaining the software for solving the problem that is relevant for you). If you are considering a product with a high donation/bugs ratio, you must consider this cost of maintenance in your cost/benefit analysis.

Probability of support in the feature

General it is needed that a software product is supported as long as it is in use. We can estimate if the OSS being evaluated will be supported in the future through an in-depth analysis of the community of OSS developers, assessing:

- The "vitality" of the product, indicated by its age and the number and frequency of releases.

- The number of companies involved in the development. A large number of companies is a good index of probability for a continuative support.
- The number of developers per company is useful to understand how important – or even "strategic"– each company considers the OSS product.
- The number of independent developers is also relevant, since a large community of developers guarantees a continuous development and maintenance effort.

4 Open BQR: a framework for the evaluation of OSS

We defined Open BQR as an extension and integration of Open BRR and QSOS to address some of the problems of the current OSS evaluation methods, which are still immature, due to the relative novelty of the field. Here, we list some problems that Open BQR helps addressing.

- Existing methods usually focus on specific aspects of OSS.
- Some methods proceed to evaluating indicators before they are weighted, so some factors may be measured or assessed even if they are later given a very low weight or even a null one. This results in unnecessary waste of time and effort.
- No OSS evaluation method adequately deals with internal and external product qualities, even though the source code is available.
- The dependence of the users of OSS is not adequately assessed, especially the availability of support over time and the cost of proprietary modules developed by third parties.

During the definition of Open BQR, we tried to build a complete, simple, repeatable, adaptable and open OSS evaluation method with the following characteristics. As such, Open BQR can be used by several types of users, including ICT experts who need to evaluate and select OSS products, OSS developers, software quality assurance and measurement professionals. The main features of Open BQR concern the investigation of a number of relevant aspects of an OSS product, including:

- Functional adequacy to requirements;
- Quality, in terms of absence of defects or time-to-fix;
- Availability of maintenance support;
- Cost of non OSS modules or necessary development tools;
- Other issues like license type, programming language ...

The evaluation process is composed of three phases, in the same line of thought as Open BRR, as we detail in the following subsections. These phases and their sub-phases consider the OSS product features outlined in Section 3.

Quick Assessment Filter

Like in the Open BRR a list of topics is identified, along with their characteristics. Unlike in the Open BRR, the characteristics are measured only after a weight has been assigned to them. The idea is to avoid data collection for characteristics that may be deemed of no or little importance, and reduce the effort and time for defining a measurement plan and collecting the data. This phase is divided into five steps, each of which addresses a different *area*, as follows.

1. **Selection of indicators based on scope and target use**: First, the application target is selected (Mission-critical, Regular, Development, Experimentation, …). Second, the license type is assessed, to check if the license type allows the development of the product as required by the specifications. Third, standard compliance, implementation language, internationalization support, books, and interests by major analysts are taken into account.
2. **Analysis of external qualities**: mainly, this step addresses the defects uncovered, the percentage of those that were fixed, and the distribution and average of the time it took to fix them.
3. **Analysis of internal qualities**: internal sub-characteristics qualities from the ISO9126 standard have been selected, along with other common indicators, such as McCabe's cyclomatic complexity.
4. **Product support over time**: this can be quantified based on the number of programmers that provide solutions to the incoming requests.
5. **Existence of required functionalities**: based on the user requirements, the functionalities of the OSS product are weighted on a 0 - 9 scale, to assess their relative importance. The required functionalities are then assessed on a 0 – 100 scale (the value '0' meaning "not implemented" and the value '100' meaning "fully implemented").

Data Collection & Processing

The outcome of the previous phase is a list of all the necessary indicators, along with their assigned weights. This phase is organized as follows:

1. **Pruning**: All of the indicators with a zero weight or a weight below a user-specified threshold are eliminated.
2. **Measurement**: The remaining characteristics are measured.
3. **Normalization**: The weights for each of the five areas described in the steps of Phase 1 are normalized to a total of 100. This allows for a fair comparison across different areas, i.e., each area will receive a score between 0 and 100.
4. **Assessment**: The final score for each area is obtained, and a single score is computed for the entire product as a weighted sum of the results obtained for the single areas, if needed. This can be useful for a first assessment, for instance for filtering out products whose total overall score is too low. An in-depth comparison among OSS products requires the knowledge of the values obtained in the single areas, along with the evaluation of costs.

Data Translation

This last phase consists in visualizing the results of the evaluation of the various products for comparison purposes. An example of a polar plot for immediate visualization is given in **Fig. 1**.

5 Validation

In order to validate the approach three well known Content Management Systems have been evaluated by means of Open BQR. The results have been compared with the informal evaluations of the same tools expressed (via forums etc.) by the community of users. For space reasons we cannot provide the entire evaluation, so we report here a few details to show how one should proceed with OpenBQR. The three products we analyzed are Mambo (http://www.mamboserver.com), Drupal (http://www.drupal.org) and WebGUI (http://www.webgui.it). We started by setting a number of requirements for the application, and we ranked them in order of importance. We envisioned a personal web application with a user-defined layout (weight 10), with user-operated functionalities for creating, reading, updating, deleting web pages (weight 10) and files (weight 5), image gallery (weight 7), and support for the Italian language (weight 4). We then carried out the other activities of the Quick Assessment Filter.

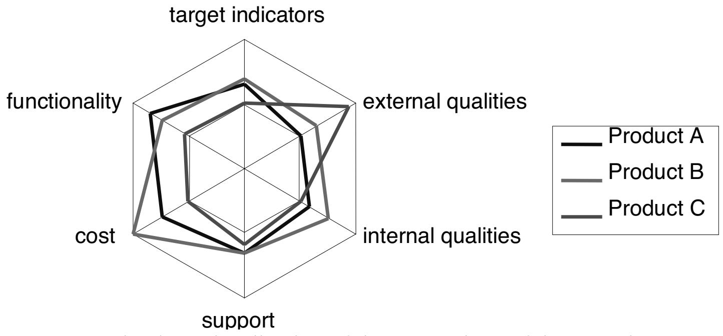


Fig. 1. Open BQR evaluation: visualization of the comparison of three products.

Some details of the evaluation are reported in Appendix A. The following comparison table reports the synthesis of the results for the three products.

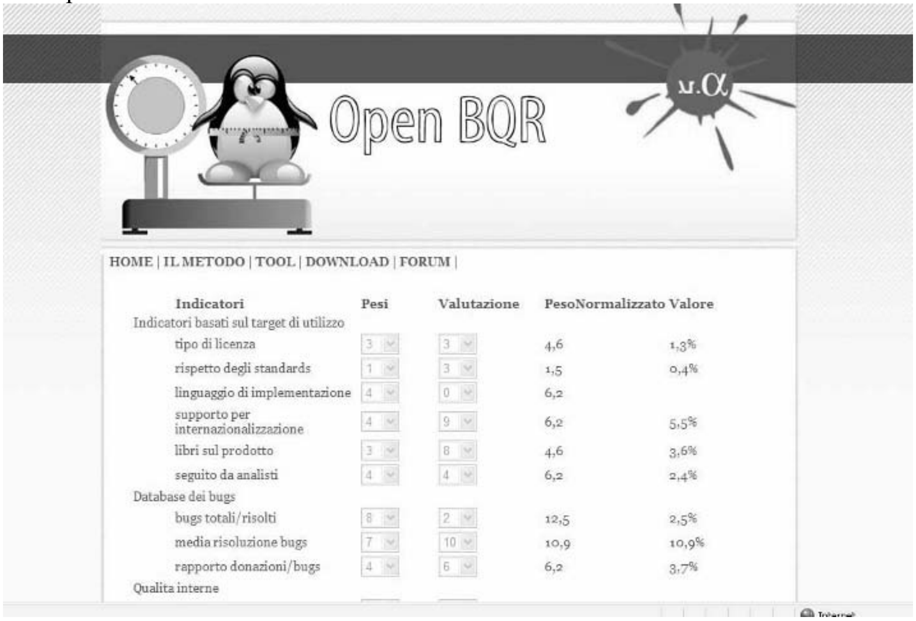
Product	**Mambo**	**Drupal**	**Web GUI**
Target usage assessment	30,69	20,31	28,45
Analysis of external qualities	11,72	9,66	14,14
Availability of support in the future	35,68	29,14	13,10
Satisfaction of functional requirements	100	100	100
Overall evaluation	**78,10%**	**68,10**	**55,69**
Rating	☆☆☆☆	☆☆☆	☆☆

According to our analysis, the best product is Mambo, although from the point of view of the external quality WebGUI is better.

6 A web-based tool implementing OpenBQR

A web based tool is being developed to help users apply the method in a coherent way. The tool is designed to be able to easily manage a complete assessment, from the requirements analysis, to the final visualization of the results. The main goal of the tool is to provide a framework for the comprehension and application of the Open BQR model, through all its steps, starting from the requirements analysis, to the assessment of all the indicators and at the end showing a complete report with the total score and a radar graph.

Through the web-based tool, we also collect data about the usage of Open BQR (what features the users consider more important, what kind of software they are interested into, etc.). These data are used to improve the method and the tool. The users' privacy is protected by a nondisclosure agreement. Data are published only in aggregated form and we take care that no information about specific users is ever made public.



Indicatori	Pesi	Valutazione	PesoNormalizzato	Valore
Indicatori basati sul target di utilizzo				
tipo di licenza	3	3	4,6	1,3%
rispetto degli standards	1	3	1,5	0,4%
linguaggio di implementazione	4	0	6,2	
supporto per internazionalizzazione	4	9	6,2	5,5%
libri sul prodotto	3	8	4,6	3,6%
seguito da analisti	4	4	6,2	2,4%
Database dei bugs				
bugs totali/risolti	8	2	12,5	2,5%
media risoluzione bugs	7	10	10,9	10,9%
rapporto donazioni/bugs	4	6	6,2	3,7%
Qualita interne				

Fig. 2. A screenshot of the Open BQR web-based tool.

7 Conclusions

In this paper, we have introduced a new OSS quality evaluation framework, which we built by integrating and extending existing approaches, so as to take advantage of their strengths, alleviate some of their drawbacks, and include some additional characteristics of interest. OpenBQR is fairly complete, simple, repeatable, adaptable, and open, so it can be used by different software organizations.

Future work will include using OpenBQR for the evaluation and comparison of OSS products in several different areas. This may lead to tailored, more specific versions of OpenBQR for the different application areas. Also, this will allow us to further validate both the way the approach is used and its usefulness in reflecting what the software industry expects from an OSS quality evaluation framework. This will entail interviews and studies that will involve all the major stakeholders, i.e., the OSS producers and the OSS users.

Acknowledgments
The research presented in this paper has been partially funded by the IST project Qualipso[9], sponsored by the EU in the 6th FP (IST-034763).

The work was also supported by the FIRB project "ARTDECO," sponsored by the Italian Ministry of Education and University, and the project "La qualità nello sviluppo software," sponsored by the Università degli Studi dell'Insubria.

8 References

[1] D. Cerri and A. Fuggetta, "Open Standards, Open Formats, and Open Source", July 2006, Submitted for publication
[2] A. Fuggetta. "Open source software: an evaluation". Journal of Systems and Software, April 2003.
[3] "Making Open Source Ready for the Enterprise: The Open Source Maturity Model", from "Succeeding with Open Source" by Bernard Golden, Addison-Wesley, 2005, available form http://www.navicasoft.com
[4] Atos Origin, "Method for Qualification and Selection of Open Source software (QSOS), version 1.6", http://www.qsos.org/download/qsos-1.6-en.pdf
[5] "Business Readiness Rating for Open Source - A Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software", BRR 2005 - RFC 1, http://www.openbrr.org.
[6] S. H. Kan, "Metrics and Models in Software Quality Engineering, 2nd Edition", Addison Wesley Professional, 2003.
[7] G. Cosenza, Liberi di Cambiare, Computer Business Review On-line Italy, http://www.cbritaly.it/Aree-tematiche/OSS/Liberi-di-cambiare. (In Italian).
[8] J. Smith David, D. Schuff, R. St. Louis, "Managing your total IT cost of ownership", Communications of the ACM, Volume 45, n. 1 (January 2002).
[9] http://www.qualipso.eu

9 Appendix A. Details of the evaluation of CMS

Product	Drupal 4.7.4	Mambo 4.5.3	WebGUI 7.0
System requirements	**Drupal**	**Mambo**	**WebGUI**
Application Server	PHP 4.3.3+	PHP 4.1.2+	mod_perl
Cost	Free	Free	Free
Database	MySQL, Postgres	MySQL	MySQL
License	GNU GPL	GNU GPL	GNU GPL
Operating System	Any	Any	Any
Programming Language	PHP	PHP	Perl
Web Server	Apache, IIS	Apache, IIS, any PHP enabled web server	Apache
Support	**Drupal**	**Mambo**	**WebGUI**
Commercial Manuals	Yes	Yes	Yes
Commercial Support	Yes	Yes	Yes
Commercial Training	Yes	Yes	Yes
Developer Community	Yes	Yes	Yes
Online Help	Yes	Yes	Yes
Public Forum	Yes	Yes	Yes
Third-Party Developers	Yes	Yes	Yes
Ease of Use	**Drupal**	**Mambo**	**WebGUI**
Mass Upload	Free Add On	No	Yes
Prototyping	No	No	Yes
Server Page Language	Yes	Yes	Yes
Spell Checker	Free Add On	No	Limited
Style Wizard	No	No	Yes
Template Language	Limited	Yes	Yes
WYSIWYG Editor	Free Add On	Yes	Yes
Built-in Applications	**Drupal**	**Mambo**	**WebGUI**
Blog	Yes	Yes	Yes
Document Management	Limited	Free Add On	Limited
File Distribution	Free Add On	Free Add On	Yes
Link Management	Free Add On	Yes	Yes
Mail Form	Free Add On	Yes	Yes
Photo Gallery	Free Add On	Free Add On	Yes

Indicators	**Assigned weight**	**Evaluation**	**Normalized weight**	**Weighted evaluation**
Target usage				
type of license	9	10	15.52	**15.52**
standard compliance	5	8	8.62	**6.90**
implementation language	0	0	0.00	**0.00**
internationalization support	4	10	6.90	**6.90**
books	2	4	3.45	**1.38**
analysts and consultants	0	0	0.00	**0.00**
External qualities				
bug number	6	8	10.34	**8.28**
average time for defect removal	4	5	6.90	**3.45**
effect of donations of defect removal speed	6	0	10.34	**0.00**
Internal qualities				
complexity	0	0	0.00	**0.00**
reuse	0	0	0.00	**0.00**
dependencies	0	0	0.00	**0.00**
others	0	0	0.00	**0.00**
Future support				
number of releases	9	10	15.52	**15.52**
number of organizations supporting the software	5	9	8.62	**7.76**
number of programmers per organization	4	9	6.90	**6.21**
number of independent programmers	4	9	6.90	**6.21**
Total			100.00	**78.10**

Network Analysis of the SourceForge.net Community

Yongqin Gao and Greg Madey

Department of Computer Science and Engineering
University of Notre Dame
{ygao1,gmadey}@nd.edu

Abstract. Software is central to the functioning of modern computer-based society. The OSS (Open Source Software) phenomenon is a novel, widely growing approach to develop both applications and infrastructure software. In this research, we studied the community network of the SourceForge.net, especially the structure and evolution of the community network, to understand the Open Source Software movement. We applied three different analyses on the network, including structure analysis, centrality analysis and path analysis. By applying these analyses, we are able to gain insights of the network development and its influence to individual developments.

1 Introduction

In recent research, network characteristics have received more and more attention, especially in evolving networks like the Internet, social networks and communication networks [22, 24, 18]. Analyzing these characteristics can reveal interesting information. In this study, we used network analysis to investigate the network characteristics in the evolution of the community network in SourceForge.net.

2 Related Work

Topology analysis is a method that can be used to understand the evolving complex networks [19, 3, 12]. It can also be used to understand the OSS phenomenon. This study also tried to understand the OSS phenomenon by studying the community as a collaboration network where every user and project can be a single node in the network.

Gao et al. [14] analyzed the empirical data they collected from SourceForge to obtain statistics and topological information of the Open Source Software developer collaboration network. They extracted the parameters and generated a model that depicts the evolution of this collaboration network. They also used these parameters to characterize the empirical data they collected from SourceForge, while other research tended to look at the network as a single snapshot in its evolution, which means they all based their observations on network, without respect to time. They were able to inspect the network with consideration of time, using the empirical data collected over more than two years.

Please use the following format when citing this chapter:

Gao, Y. and Madey, G., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 187–200.

Xu, Madey and Gao [25] presented the results of docking [9] a Repast [17] simulation and a Java/Swarm [16] simulation of four social network models of the Open Source Software community. The simulations grew "artificial societies" representing the SourceForge developer/project community. As a byproduct of the docking experiment, they provided observations on the advantages and disadvantages of the two toolkits for modeling such systems.

These previous analyses studied the OSS community based on the global topology of the collaboration network. These methods were not capable of revealing behaviors of a single object such as a user or a project. Our study extended the understanding of OSS to the study of individual behaviors and introduced a new measure set in the study of the OSS community.

3 Our Approach

The analysis we used includes structure analysis, centrality analysis and path analysis. We conducted the analyses in the following manner. First, we conducted the structural analysis, including the following measures: diameter, clustering coefficient and component distribution. Then we conducted the centrality analysis, including the following measures: average degree, degree distribution, average betweenness and average closeness. Finally, we conducted the path analysis on most of the previous measures.

3.1 Structure Analysis

The first analysis is the structure analysis [20, 10]. Structure analysis is used to inspect the macro-measures of the network structure. The measures inspected in the structure analysis describe the network structure in a global view. Study of these measures helps us understanding the influence of network structure to individual nodes in the network.

The *diameter* of a network is the maximum distance (number of hops or edges) between any pair of nodes. The diameter can also be defined as the average length of the shortest paths between any pair of nodes in the network. In our research, we are more interested in the measures that can describe the efficiency of information propagation. So the average value is more suitable for our purpose, and we used the second definition in our research. Strictly speaking, the diameter of a disconnected graph (i.e., one containing isolated components) is infinite, but it is normally defined as the maximum diameter of its sub-clusters or other approximate values. Random graphs and other complex networks all tend to have small diameters. This is the phenomenon scientists referred to as the "small world phenomenon" [23]. The smaller the diameter of a network is, the better the network is connected. The diameter is one of the important attributes in complex network research, especially since the small world phenomenon[1] was popularized. We calculated the diameter measures using approximate method, which can generate fairly accurate results, especially when the network size is huge ($N > 10,000$). More detailed explanation and discussion can be found in [13].

[1] "Six degrees of separation" is a famous claim by Ouisa, a popular character in John Guare's play (1990)

The equation we used to calculate the approximate diameter D is

$$D = \frac{log(N/z_1)}{log(z_2/z_1)} + 1 \tag{1}$$

where N is the number of nodes in the network, z_1 is the average degree of nodes in the network, and z_2 is the average number of nodes two steps away from a given node as defined in [13].

The next measure is *clustering coefficient*. The neighborhood of a node consists of the set of nodes to which it is connected. The clustering coefficient of a node is the ratio of the number of links to the total possible number of links among the nodes in its neighborhood. The clustering coefficient of a graph is the average of the clustering coefficients of all the nodes. Recent research has found that real complex networks typically have a high clustering coefficient, which means that they exhibit a large degree of clustering [5]. Clustering coefficients of some real networks, such as the network we studied in SourceForge, can be calculated more easily from related bipartite graphs [21] by using the generating function method for bipartite graphs. More detailed explanation of this method can be found in [13].

Using this method, the clustering coefficients of these kinds of bipartite structures result in a non-vanishing value,

$$C = \frac{1}{1 + \frac{(\mu_2 - \mu_1)(\nu_2 - \nu_1)^2}{\mu_1 \nu_1 (2\nu_1 - 3\nu_2 + \nu_3)}} \tag{2}$$

where $\mu_n = \sum_k k^n P_d(k)$ and $\nu_n = \sum_k k^n P_p(k)$. In the developer-project bipartite network, $P_d(k)$ represents the fraction of developers who joined k projects, while $P_p(k)$ means the fraction of projects that have k developers.

The last measure in the structure analysis is the *component distribution*. A component of a network is defined as the maximal subset of connected nodes. To formalize the definition of a component, first we define a path in a network as:

– A path $v_1 e_1 v_2 ... e_{n-1} v_n$ is a sequence of nodes such that from each of its nodes v_i there is an edge e_i to the next node v_{i+1} in the sequence. Normally, the first node v_1 is called the start node and the last node v_n is called the end node.

Then the component C of a network can be defined as:

– Component C is a subset of (V,E) of a network. For any pair of nodes v_i and v_j, where $v_i, v_j \in C$, there exists a path $v_i e_i ... e_{j-1} v_j$ between these two nodes. And for any any pair of nodes v_k and v_l, where $v_k \in C$ and $v_l \notin C$, there doesn't exist a path $v_k e_i ... e_{j-1} v_l$ between these two nodes.

3.2 Centrality Analysis

The second analysis is the centrality analysis. Centrality analysis is used to inspect the micro-measures of the network structure or the relative importance of a node within a network. Study of these measures helped us understand the influence of individual nodes to the global network structure.

The first measure is *degree*. The degree of a node, k, equals the total number of other nodes to which it is connected, while $P(k)$ is the distribution of the degree k throughout the network. Degree distribution in real networks was believed to be a normal distribution (when $N \rightarrow \infty$), but recently, Albert and Barabási and others found it fit a power law distribution in many real networks [7]. The other measure related to degree is the average degree as $\sum P(k)/N$, which is the average of the node degrees in the network.

The next measure is *betweenness*. Betweenness is a centrality measure of a node within a network. Nodes that occur on many shortest paths between other nodes have higher betweenness than those that do not. For a graph $G(V, E)$ with n nodes, the betweenness $B(v)$ for node v is

$$B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{3}$$

where σ_{st} is the number of shortest geodesic paths from s to t, and $\sigma_{st}(v)$ the number of shortest geodesic paths from s to t that pass through the vertex v. This may be normalized by dividing through by the number of pairs of vertices not including v, which is $(n-1)(n-2)$.

The last measure is *closeness*. Closeness is also a centrality measure of a node within a network. Nodes that are "shallow" to the other nodes (that is, those that tend to have short geodesic distances to other nodes within the network) have higher closeness. Closeness is preferred in centrality analysis to mean shortest-path length, as it gives higher values to more central nodes, and so is usually positively associated with other measures such as degree.

The closeness $C(v)$ for a vertex v is the reciprocal of the sum of geodesic distances to all other vertices in the graph:

$$C(v) = \frac{1}{\sum_{t \in V} d_G(v, t)}. \tag{4}$$

3.3 Path Analysis

All the previous analyses (structure analysis and centrality analysis) are based on network snapshot topology. They depict the characteristics of a static network at a given point of time. But these are not the only important analyses in a network, especially an evolving network. With sequence of network snapshots instead of just single snapshot of the networks, we are able to inspect not only the measures (diameter, clustering coefficient, component, degree, betweenness and closeness) in the previous analysis, but also the developing trends of these measures.

We conducted the path analysis on the diameter, clustering coefficient, average degree, betweenness and closeness. By inspecting these developing measures, we are looking forward to understanding more about the life cycle of the network and individual nodes in the network.

4 Results and Discussion

Before discussing these analyses and measures, we need to explain the collaboration network that we studied. From SourceForge.net, we got data on two major entities – developers and projects [2]. In this data, only one relationship existed – the participation between developer and project. There were no direct links between developers and between projects. So, we looked at this network as a bipartite network, where projects and developers were the two kinds of nodes, and edges could only connect different kinds of nodes. There are two transformations from this network – the developer network and the project network. In the developer network, there is only one type of node representing the developer in the collaboration network, and the edge in the network represents the relationship of collaboration. For every pair of nodes i and j, there is an edge connecting i and j only if i and j are collaborating on at least one project. We also generated the project network, where a node represents a project in the collaboration network and the edge in the network represents the relationship of sharing the same developer(s). In the following discussion, we will abbreviate these three networks as P-NET(project network), D-NET(developer network) and C-NET(collaboration network).

4.1 Structure Analysis

The first analysis we applied is the structure analysis, including measures such as diameter, clustering coefficient and component distribution. As discussed in the previous section, the diameter is approximate on the whole network by the equation 1. The resulting approximate diameters for the D-NET are between 5 and 7, while the number of developers in the D-NET ranged from 97,705 to 123,968. Thus, the diameter of the network is quite small with regard to the overall network size (the number of developers in the network). On the other hand, the approximate diameters for the P-NET are between 6 and 8, while the number of projects in the P-NET ranged from 70,089 to 91,713. So the diameter is relatively stable compared to the significant increase of the network size.

The next measure is the clustering coefficient. We also used the approximate clustering coefficient by applying the equation 2. The resulting approximate clustering coefficients for the D-NET are between 0.85 and 0.95. On the other hand, the approximate clustering coefficients for the P-NET are between 0.65 and 0.75. High clustering coefficients reveal the highly clustered property of both the D-NET and the P-NET, which is similar to the results we got from our previous study conducted in, although the networks have been expanded significantly.

Both diameter and clustering coefficient are popular and efficient measures to describe the structure property of a network, especially the cluster property. Highly clustered networks are normally favored in real evolving complex networks like communication networks or social networks for better information propagation.

From the previous measures, we understand that both D-NET and P-NET are highly clustered networks. But these measures do not mean the networks are fully connected. Actually, most of the real networks are not fully connected. There will be

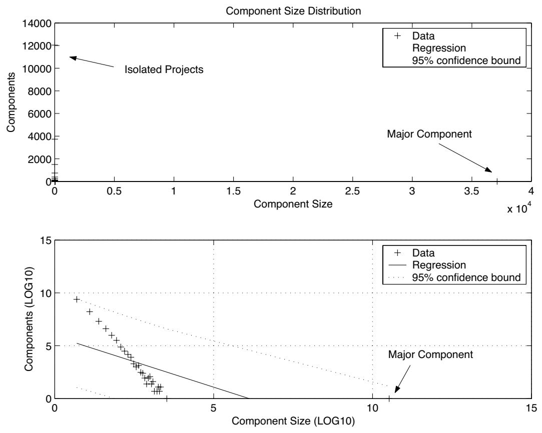


Fig. 1. Project Network Component Distribution

connected parts in the network, which we called as "component". The next measure we inspected is the component distribution. In the SourceForge community, power law exists in the component distribution of the networks. In Figure 1, the component distribution for the P-NET for June 2006 is shown. There are two figures. In the lower figure, after applying log transformation on both coordinates, we found that the component distribution fits a straight line quite well without considering the biggest component (which will be called the major component in latter discussion). The R^2 [11] of linear regression with the major component is 0.4023 and the R^2 of linear regression without the major component is 0.9886. Also, in the lower figure, we illustrated the 95% confident boundary for the linear regression as the dot line. In the upper figure, where the coordinates are in normal scale, we made another interesting discovery: almost all the components are quite close to each other, except the two extremes. One extreme is the major component and the other is the isolated components, which include only isolated developers.

4.2 Centrality Analysis

The second analysis we conducted is the centrality analysis, which focus on the following measures – degree, betweenness and closeness.

Degree is the simplest measure of the connectivity of a node in the network. We also used the C-NET, D-NET and P-NET for June 2006 as examples in this section. There are totally four degrees of developer and project in these three networks:

– *Degree of developer in the C-NET* is the number of projects in which a developer participated in the community.
– *Degree of developer in the D-NET* is the number of developers who have at least one collaboration with the given developer in the community.

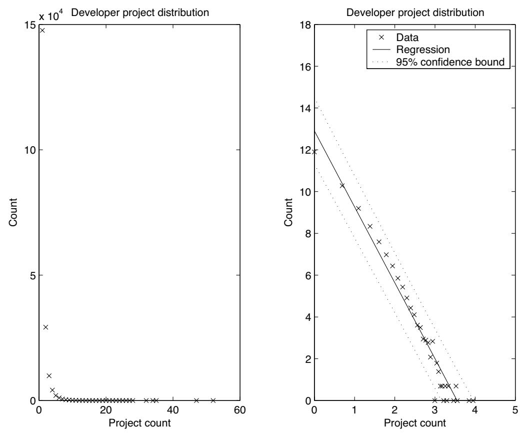


Fig. 2. Developer Size Distribution

– *Degree of project in the C-NET* is the number of developers who participated in the given project in the community.
– *Degree of project in the P-NET* is the number of projects share by at least one common developer with the given project in the community.

In the June 2006 dataset, the average degree of developer in the C-NET is 1.4525; the average degree of developer in the D-NET is 12.31; the average degree of project in the C-NET is 1.7572; the average degree of project in the P-NET is 3.8059, while the sizes of the C-NET, the D-NET and the P-NET are 215,681, 123,968 and 91,713. The average degree of developer and project in the C-NET is relatively low since the isolated developer (developer with single project) and the isolated project (project with only one developer) are big parts of the community.

Then we investigated the degree distributions of the SourceForge.net community. Degree distribution is proven to have a normal distribution in the ER model when $N \rightarrow \infty$. This was believed to be a good model for the real complex network before the power law was reported for many real network systems by Barabási et. al [6]. In the SourceForge community, we found that the degree distributions (distributions for all four degrees) also followed power law. Figure 2 and Figure 3 show two of the degree distributions.

These figures are based on the dataset from SourceForge.net for June 2006. The left figures are the degree distributions in normal coordinates. To verify the existence of power law in these distributions, we applied log-log transformations on the data to generate the right figures, which are the degree distributions on log-log coordinates. The 95% confident boundaries for the linear regression also are provided on the figures for all the log-log transformed degree distributions. The R^2 of linear regression for the developer degree distribution in the C-NET is 0.9577. The R^2 of linear regression for

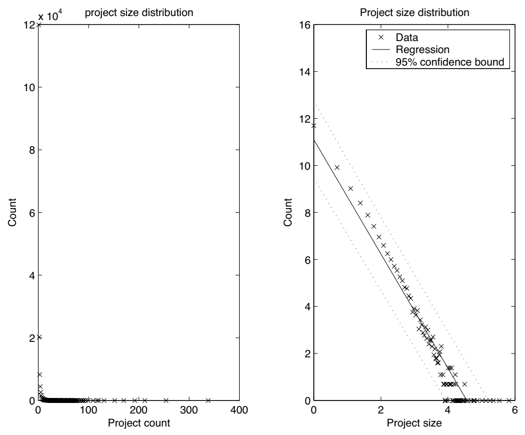


Fig. 3. Project Size Distribution

the project degree distribution in the C-NET is 0.9173. Thus, these distributions fit power law distributions very well.

Betweenness and closeness are also the common measures for centrality analysis. Betweenness is a measure to describe the importance of the node in the network according to shortest path, and closeness is a measure to describe how close the node is to other nodes. Betweenness is a normalized value in $[0, 1]$. The higher the measure is, the more central the node is to the network. Closeness is also bounded by $[0, 1]$, but it is not normalized. So closeness tends to decrease when the network size is increasing. Normally, these measures yield very small value in large networks ($N > 10,000$), so comparison of these measures only makes sense when comparing networks of similar size. Also, using the dataset from SourceForge.net for June 2006, the average betweenness for the P-NET is 0.2669e-003 and the average closeness for the P-NET is 0.4143e-005, which are relatively large for a network this size.

4.3 Path Analysis

All the measures in the previous sections (diameter, clustering coefficient, component, degree, betweenness and closeness) are about the topology of the networks. They depict the characteristics of a static network at a given point of time. These are not the only important attributes in a network, especially an evolving network [15, 4]. Since we had multiple monthly database dumps from SourceForge.net, we were able to investigate the development patterns of these measures of the networks. By applying the path analysis, we can study the life cycle and the evolving patterns of the network and individuals in the network.

The first path analysis is on the average degrees. Average degree $< k >$, which gives the average number of links per node, is a good quantitative measurement for the connectivity of a graph. Two of the developing pattern of the average degrees are

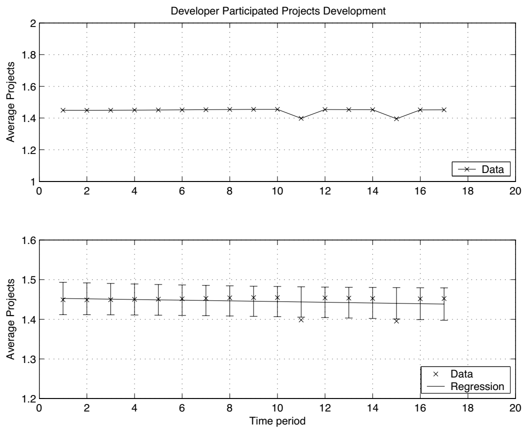


Fig. 4. Average Developer Degree in the C-NET

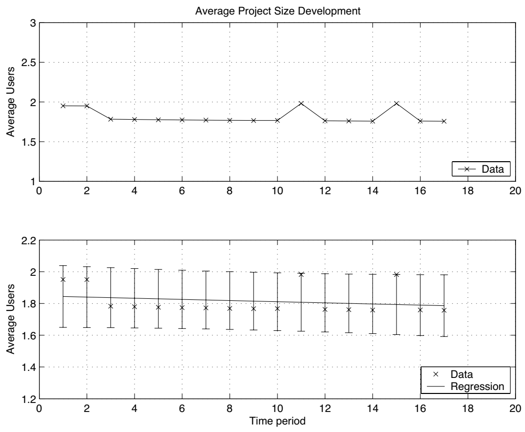


Fig. 5. Average Project Degree in the C-NET

shown in Figure 4, Figure 5. The X coordinate in the figure is the number of months that passed after February 2005.

For the average degrees, we show the linear regression in the lower figures with a 95% error bar for every data point. The slope of the regression for developer degree is -0.0009 and the slope of the regression for project degree is -0.0036. The average degrees are actually decreasing, which means the average project size and average number of projects a single developer participated in are decreasing.

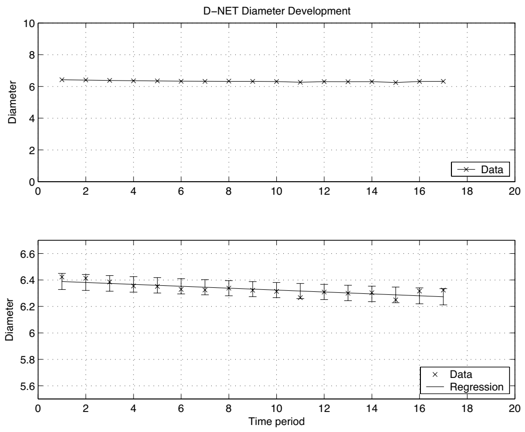


Fig. 6. Developer Network Diameter

Diameter and clustering coefficient is closely related to average degree. We will conduct path analysis on these two measures next. In this paper, we discuss only the D-NET, detailed discussion about other networks can be found in [13].

The diameter of the D-NET is a good measure of network communication ability. A shorter diameter results in fewer average steps needed for one developer to spread a message to another developer and less time needed for an idea to spread through the network. The D-NET has a small diameter, which was calculated in a previous section. Also, we investigated the evolution of the diameter of the D-NET, as shown in Figure 6.

The figure indicates that D decreases with time, which is different from the previous research [8] on random networks that reports that diameter increases with network size. The lower figure shows the linear regression with 95% error bar for the developing trend of diameter for the D-NET. The slope of the regresion is -0.0072.

Clustering coefficient is another important measures of the topology of real networks. So the clustering coefficient, a quantitative measure of clustering, CC, is also a measure we investigated. The approximate clustering coefficient for the D-NET as a function of time is shown in Figure 7.

In the figure, we can observe the increasing trend of the clustering coefficient. The lower figure shows the linear regression with 95% error bar for the developing trend of clustering coefficient for the D-NET. The slope of the regression is 0.0011. The clustering coefficient for the D-NET tells us how much a node's co-developers are willing to collaborate with each other, and it represents the probability that two of its developers are collaborating on a project. Thus, with the evolution of the D-NET, more edges (collaboration relations) are formed. This will lead to an increase in the connectivity of the developer with the neighboring developers. Furthermore, this leads to the increase in the clustering coefficient.

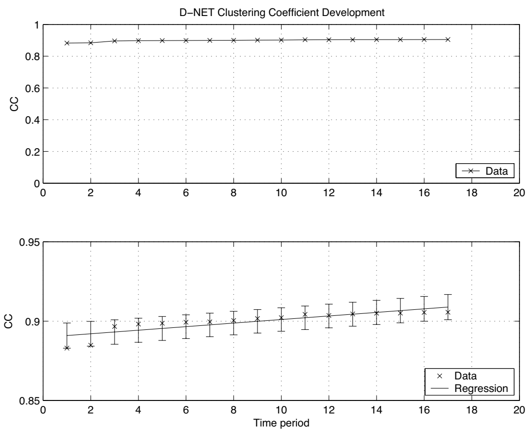


Fig. 7. Developer Network Clustering Coefficient

Increases on clustering coefficients in D-NET suggest the network is evolving to improve its cluster property. The higher the clustering coefficient is, the more connected the network is.

Now we will apply the path analysis on average betweenness and average closeness. Figure 8 shows the developing trend of average betweenness in the P-NET. In the upper figure, we can observe the almost flat developing trend of the betweenness, although the overall size of the network has increased significantly during the same time period. This can suggest that the network is in a stable topology in its own evolution. In the lower figure, we magnify the Y coordinate to have a close look at the developing trend of the average betweenness. The average betweenness has a slightly increasing trend. This observation can be explained by the "rich get richer" phenomenon discovered in other complex networks such as the Internet. Although the network is constantly expanding, the hub (the node with most links) will keep gaining more connections than the others. Also, alternative hubs or regional hubs will also emerge from the network, and these hubs will increase the average betweenness of the network.

Average closeness is another measure of centrality. Figure 9 shows the developing trend of the average closeness in the P-NET. The upper figure is the developing trend in normal coordinates and the lower figure is the developing trend in magnified Y coordinates. We can observe the similar developing behavior of the average closeness to the average betweenness. But average closeness is more flat than the average betweenness. This is because closeness for individual node is not normalized like the betweenness for individual node. Thus, the significant increase in the overall network size will have more influence on the closeness than on the betweenness. Therefore the developing trend of the average closeness will be more flat than the developing trend of the average betweenness.

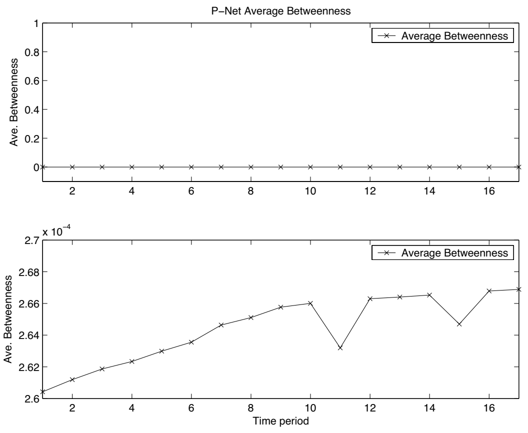


Fig. 8. Project Network Betweenness

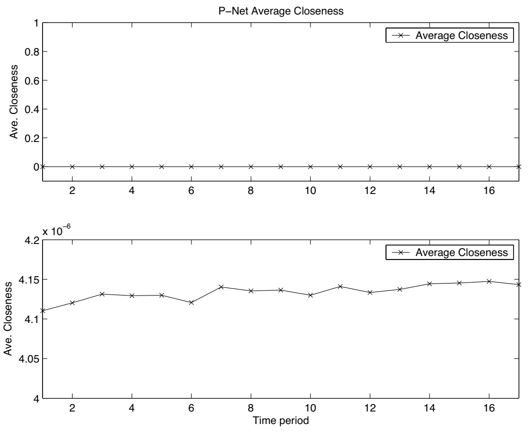


Fig. 9. Project Network Closeness

By using path analysis, we are able to look at not only the topology of the network at a given time, but also at the evolution of the network topology, and the mutual inferences between a single entity in the network and the whole network.

One of the common discoveries in all the path analyses is life-cycle like behaviors. Most of the measures have sustained a stable level throughout the inspected period of time in this study and also have increased/decreased from the previous study carried out in [13]. This phenomenon suggests that we have witnessed the beginning of the evolution of the SourceForge.net community and possibly the mature (stable) era of the SourceForge.net community. By closely watching the SourceForge.net community, we may have deeper insight into the evolution of the OSS community. Also, similar analyses can be applied to other evolving complex networks, such as communication networks or social networks to study the life cycle of those networks.

We also have made another discovery about the network measures. We observed a strong tie between the network measures and the evolution of the community networks. This discovery suggests that the network measures can be used not only as a predictor of the future of an individual entity in the network or the whole network.

5 Conclusion

In this study, we studied multiple network measures of the SourceForge.net community network and their evolution patterns by applying multiple analyses, including structure analysis, centrality analysis and path analysis. In the structure analysis, we calculated the diameter, clustering coefficient and component distribution. The two approximate methods used to calculate the approximate diameter D and approximate clustering coefficient CC are

$$D = \frac{log(N/z_1)}{log(z_2/z_1)} + 1$$

$$CC = \frac{1}{1 + \frac{(\mu_2-\mu_1)(\nu_2-\nu_1)^2}{\mu_1\nu_1(2\nu_1-3\nu_2+\nu_3)}}.$$

In the centrality analysis, we calculated four different average degrees and four different degree distributions. Also, we calculated average betweenness and average closeness. The equations to calculate the betweenness $B(v)$ and closeness $C(v)$ for individual nodes are

$$B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

$$C(v) = \frac{1}{\sum_{t \in V} d_G(v,t)}.$$

In the path analysis, we investigated developments of the multiple measures, including average degrees, diameter, clustering coefficient, average betweenness and average closeness.

Acknowledgment. This work was supported in part by the National Science Foundation, CISE/IIS-Digital Society & Technology, under Grant No. 0222829 [1].

References

1. OSS research (http://www.nd.edu/˜oss). 2006.
2. OSS research portal (http://zerlot.cse.nd.edu). 2006.

3. L.A. Adamic and B.A. Huberman. Scaling behavior of the world wide web. *Science*, 287(2115), 2000.
4. J. Ahola. Mining sequential patterns. *VTT research report TTE1-2001-10*, 2001.
5. R. Albert and A.L. Barabási. Dynamics of complex systems: Scaling laws for the period of boolean networks. *Physics Review*, 85(5234), 2000.
6. R. Albert and A.L. Barabási. Statistical mechanics of compex networks. *Reviews of Modern Physics*, 74(47), 2002.
7. R. Albert, H. Jeong and A.L. Barabási. Diameter of world-wide web. *Nature*, 401(130), 1999.
8. B. Bollobás. Random graphs. *London:Academic*, 1985.
9. R. Burton. *Simulating Organizations: Computational Models of Institutions a nd Groups*, chapter Aligning Simulation Models: A Case Study and Results. AAAI/MIT Press, Cambridge, Massachusetts, 1998.
10. J. Camacho, R. Guimerà and L.A.N. Amaral. Preprint cond-mat/0102127. 2001.
11. Online document. Pearson's r. http : //www.analytics.washington.edu/ rossini/courses/intro − nonpart/text/Pearson_s_l_r_l_.html.
12. M. Faloutsos, P. Faloutsos and C. Faloutsos. On power-law relationships of the internet topology. *Computer Communication Review*, 29(251), 1999.
13. Y. Gao. Topology and evolution of the open source software community. *Master Thesis*, 2003.
14. Y. Gao, V. Freeh and G. Madey. Analysis and modeling of the open source software community. *NAACSOS, Pittsburgh*, 2003.
15. J. Hamilton. Time series analysis. *Princeton University Press, Princeton, NJ*, 1994.
16. D. Hiebeler. The swarm simulation system and individual-based modeling. *Advanced technology for natural resource management*, 1994.
17. Repast Information. http://repast.sourceforge.net/. 2002.
18. F. Liljeros, C. Edling, A. Lan, S. He and Y. Aberg. Hub caps could squash STDs. *Nature*, 411(907), 2001.
19. A. Border, R. Kumar, F. Maghoul, P. Raghavan, S. Rajalopagan, R. Stata, A. Tomkins and J. Wiener. Graph structure in the web: experiments and models. *Computer Networks*, 33(309), 2000.
20. J.M. Montoya and R.V. Solé. Preprint cond-mat/0011195. 2000.
21. M.E.J. Newman and D.J. Watts. Random graph models of social networks. *Physics Review*, 64(026118), 2001.
22. S.L. Pimm. *The Balance of Nature*. University of Chicago Press, Chicago, 1991.
23. D.J. Watts. *Small world.* Princenton university press, NJ, 1999.
24. R.J. Williams, N.D. Martinez, E.L. Berlow, J.A. Dunne and A.L. Barabási. Two degrees of separation in complex food webs. *Santa Fe Institute Working Paper Series*, (01-07-036), 2001.
25. J. Xu, Y. Gao, J. Goett, and G. Madey. A multi-model docking experiment of dynamic social network simulations. *Agent conference, Chicago, IL*, 2003.

Part II

Short Papers

Influence in the Linux Kernel Community

Timo Aaltonen and Jyke Jokinen
Institute of Software Systems, Tampere University of Technology
first.last@tut.fi

Abstract. Several success stories of open source (OS) products have been seen during last decade. Due to the economical importance of the products, it is important to know who are the ones who have the largest influence to the products. Is there a dominant player in developing communities? In this paper[1] the aspect is studied with respect to the Linux Kernel community. We show that the influence is centered to a small number of core people, and corporates have a large impact to the development. Moreover, we enumerate the most influential companies.

Key words: data mining, Linux kernel

1 Introduction

Open source (OS) software development has gained much attention lately. During last decade several success stories, like Apache, Mozilla and Linux, has been seen. Apache is the market leader of the world's web servers [2] having over three times the market share of its next-ranked (proprietary) competitor. Internet Explorer has been losing market share to OS web browser, especially to Mozilla [3]. Linux [4] is a free UNIX-type operating system originally created by Linus Torvalds.

Due to the economical importance of open source, it is important to know, who influences the development. Is it carried out by altruistic individuals and what is the impact of large organizations? By knowing these facts one is able to predict the directions how the products evolve in future. This is essential when choosing between different open source and proprietary alternatives.

This paper studies the influence of the developers and leaders of the Linux Kernel. The Kernel was chosen because it is the only operating system challenging Microsoft Windows, the available amount of data is large, and the number of people working for the project is numerous.

The study of influence is based on counting the signers of *Developer's Certificate of Origin* [5] (DCO) for patches. In short, signing a DCO has two main meanings

[1] This paper is a revised version of [1]

1. the original author of a patch certifies that she has the right to submit it under the open source license indicated in the file; and
2. later code maintainers and Linux lieutenants indicate that they accept the patch by adding their own signature.

It is obvious that the signers are influential persons in the Linux Kernel community. The more person signs patches the more influence she has.

We applied a set of measures to the mined data. The most important findings are

1. A large portion of the influence is contributed by a relatively small amount of people.
2. Based on studies on e-mail addresses, corporations seem to have much influence in the Linux Kernel community.
3. The most influential companies can be studied by relating the most influential persons to their employees, and summing up the number of signed DCOs for each company.

We have mined our data from publicly available sources. The DCO signatures have been mined from GIT [6] revision control system used by Linux Kernel developers. Technical details of the GIT data mining are presented in [1]. Personal data of the signers have been searched with Google and from certain public data sources. Section 1 introduces the measures, which are applied to individuals in Section 2 and to companies in Section 3. Section 4 discusses the paper.

2 Measures

We have developed a set of measures to be applied to our data. The measures are divided into two categories: *personal*, *company-related*. The personal measures attempt to highlight various aspects of people in the Linux Kernel community:

– **Influence distribution.** Number of signed patches are counted for each person. Then these (person, amount) pairs are sorted in descending order. The measure illustrates how the control and development work is distributed in the community.
– **E-Mail domain distribution.** The Linux Kernel development is highly geographically distributed. This measure shows where and by which kind of organizations does the decision-making takes place.
– **E-Mail taxonomical distribution.** Measure attaches a category to e-mails from taxonomy: corporate, open source project, ISP, e-mail provider, university, personal domain, and other.

The company-related measures attempt to reflect the role of companies in the development:

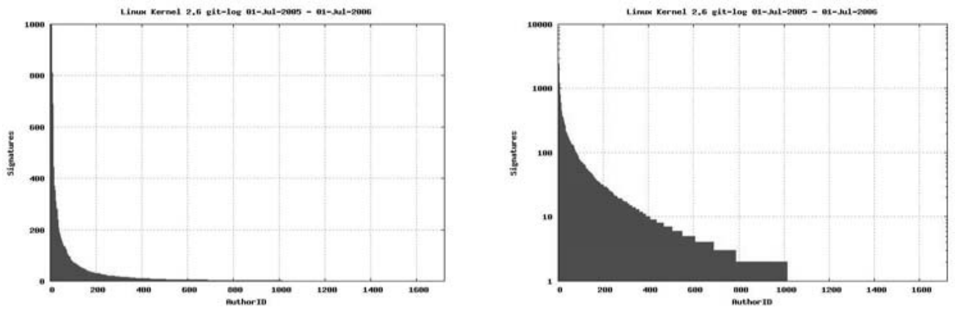
– **Impact of Companies.** Leaders and developers of the Linux Kernel community signing the patches are related to companies they work for. Then the influence of employers of each company are summed together. This sum is the influence of the company.

The measures *E-Mail taxonomical distribution* and *Impact of Companie*s are the most interesting (and the most controversial) measures. The interesting piece of information they reveal is the role of companies in the Linux community. The former indicates what is the impact of companies together, and the latter shows which individual companies are the most influential. Both measures are controversial, since their evaluation is based on opinion and skills of the researcher(s) evaluating them. Another evaluators might get slightly different values. However, we believe, that the measures describe interesting aspects of the Kernel community, even if their accuracy is not ideal.

33 Measures for Individuals

The influence distribution of the Linux Kernel developers is depicted in Figure 1. The number of signed patches is on the y-axis and individual signers are on the x-axis sorted with respect to the number of sign-offs.

A notable shape of the curve slanting to the left is quite common in open source projects. Actually, the y-axis has been truncated to make the shape of the curve more visible. The curve takes this shape because a small number of core people lead the whole community. In our previous studies [7] we have noticed that a small group of developers contribute more than the rest of the group. We call this phenomenon the *flagpole effect.*



(a) linear

(b) logarithmic

Fig. 1. Influence distributions in linear and logarithmic scales.

To make clearer the strength of the flagpole effect, the influence distribution is redrawn on a logarithmic scale in Figure 1b. It is somewhat surprising, that even now, the curve tends to slant to the left so heavily.

E-Mail Domain Distribution shows that the Linux Kernel development is highly distributed. The measure is based on studying the e-mail addresses of the persons who sign the patches. Figure 2 illustrates the distribution with respect to highest level domains. Not surprisingly, *com* domain is the number one in this measure. The second place is taken by *org*, and the third one is occupied by*de* domain, implying that many of the Kernel developers are from Germany.

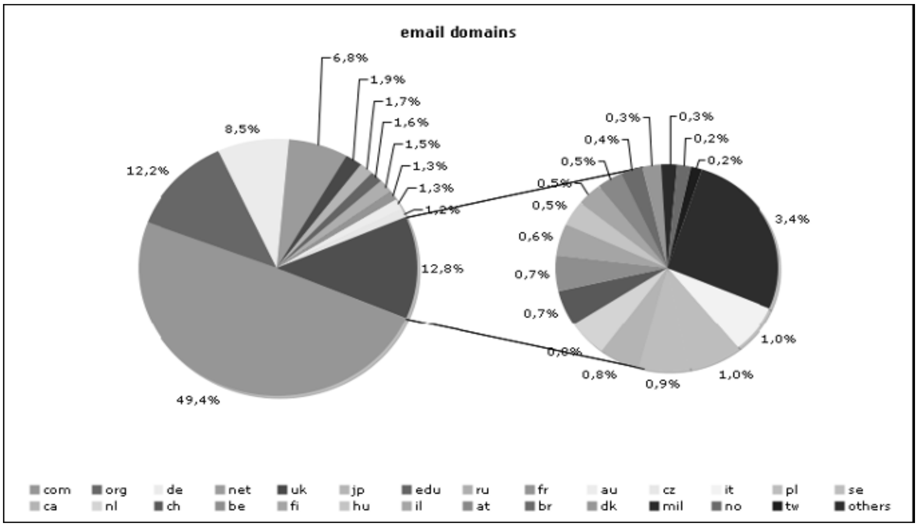


Fig. 2. The e-mail domains of patch

E-Mail Taxonomical Distribution was made by attaching each domain a category from taxonomy: *corporate*, *open source project*, *ISP*, *e-mail provider*, *university*, *personal domain*, and *other*. Google was used manually to attach a category to the domain. The results are illustrated in Figure 3. The distribution has one unexpected result: category *personal domain* taking the second place is somewhat surprising.

Category	Number	Category	Number
corporate	342	ISP	110
personal domain	207	open source project	78
other	200	e-mail provider	21
university	114		

Fig. 3. The taxonomy of e-mail addresses

4 Measures for Companies

We took a closer at the top 100 signers according to criterium of most signed patches, and used Google search engine to study whether the top 100 leaders were employed by some organization. Then we were able to calculate the size of the impact of the organizations to the Linux Kernel development.

The search techniques we used were various. We had two obvious starting points: a name and an e-mail address. If a developer had a company-related e-mail then it is quite obvious that she works for the company. Few developers had their CV on www, which was easy to find with a simple search. Book publishers and organizer of open-source-related conferences maintain lists of their contributors with a small description of people's careers on www. Often, these people were among the top 100 leaders to the Linux Kernel. One surprisingly fruitful technique was search with the name part from an e-mail address. People seem to preserve their original e-mail names in their e-mail addresses. This way the employer was joined to a set of contributors. Some people were found from Wikipedia [8]. Moreover, several creative searches were carried out.

The results of **Impact of Companies** measure are shown in Figure 4. The company with the largest impact during our time interval has been SteelEye Technology. Actually, all 928 signatures related to the company have been signed by a single person. Obviously SteelEye Technology has been very active during our time window, and perhaps all patches from the company are signed by the person. After SteelEye Technology, the next companies should not be a surprise. Google's rank has been improved by Andrew Morton's migration to the company [9].

5 Discussion

We studied the Linux Kernel development by mining data from GIT repository, and applying four measures to the mined data. The measurements show that relatively small amount of people control the development. Similar results have been reported earlier in [7]. E-Mail domain distribution and taxonomical distribution show that the Linux Kernel is mostly developed in western countries and corporations have half of the influence. Most of the most influential companies are quite expected. However, the most influential company being SteelEye Technology was not expected result.

Similar research to ours has been published in [10]. In this study patches between version 2.6.19 and 2.6.20 of Linux Kernel has been analyzed (our study used a one year time frame between July 1st 2005 and July 1st 2006). The study contains more measures than our study, and they are partly the same (Influence distribution). Similarly to us, the people have been joined to the companies they work for, and this information has been used to compute measures for companies. However, the joining method is simpler than ours, since it is based only to the email domains, whereas we have used more sophisticated searches.

Company	impact	Company	impact
SteelEye Technology	928	Oracle	136
IBM	924	Symantec	135
Google	759	Academic (all universities)	135
Intel	742	MISC	133
Novell	665	Broadcom Deep Blue	131
OSDL	588	Solutions Limited	121
UNKNOWN	453	Qlogic	114
Cicso (Topspin)	421	CoopTel	107
		MontaVista Software	105
Debian	376	Freescale	98
Alcatel	322	Hewlett-Packard	94
Red Hat	302	Network Appliance	92
Netfilter (a project)	293	Circle Computer Resources	86
Linutronix	283	Mellanox Technologies	85
Conectiva	280	Ultra	79
Ameritech	260	Toshiba	77
Dunvegan Media	184	Motorola	74
Simtec Electronics	165		
Wise Riddles Software	164		
SGI	155		
Levanta (previously Linuxcare)	138		

Fig. 4. Companies and the number of patches signed by the personnel.

Acknowledgements

This work has been partly funded by Academy of Finland, project Managing Open Source Software as an Integrated Part of Business (OSSI).

References

1. T. Aaltonen and J. Jokinen, “Demography of linux kernel developers,” Tech. Rep. 41, Tampere University of Technology, Institute of Software Systems, 2006.
2. Netcraft, “Web server survey.” http://news.netcraft.com/archives/web_server_survey.html/, 2006.
3. R. McMillan, “Mozilla gains on IE,” *PC World*, 2004.
4. “Linux online.” http://linux.org, 2006.
5. L. Torvalds, “Developer’s certificate of origin 1.1.” http://www.osdl.org/newsroom/press_releases/2004/2004_05_24_dco.html, 2006.
6. L. Torvalds and J. C. Hamano, “GIT -fast version control system.” http://git. or.cz, 2006.
7. T. Aaltonen, J. J¬arvenp¬a¬a, and T. Mikkonen, “Oss architecture and implications,” tech. rep., eBRC, 2006.
8. “Wikipedia, the free encyclopedia.” http://en.wikipedia.org/wiki/Main_ Page/, 2006.
9. B. Pro_tt, “Morton gets googled,” *Linux Today*, 2006.
10. Corbet, “Who wrote 2.6.20?.” http://lwn.net/Articles/222773/, February 2007.

EDOS Distribution System: a P2P architecture for open-source content dissemination

Serge Abiteboul[1], Itay Dar[2], Radu Pop[3], Gabriel Vasile[1] and Dan Vodislav[4]

1. INRIA Futurs, France {firstname.lastname}@inria.fr
2. Tel Aviv University, daritay@post.tau.ac.il
3. INRIA-Mandriva, Paris, France radu.pop@inria.fr
4. CEDRIC-CNAM Paris, France vodislav@cnam.fr

Abstract. The open-source software communities currently face an increasing complexity of managing the software content among theirs developers and contributors. This is mainly due to the continuously growing size of the software, the high frequency of the updates, and the heterogeneity of the participants. We propose a distribution system that tackles two main issues in the software content management: efficient content dissemination through a P2P system architecture, and advanced information system capabilities, using a distributed index for resource location.

1 Introduction

Faced with the increasing need of sharing, retrieving and loading data on the web, the problem of distributing content to large communities across the web has acquired a growing importance.

In the particular case of open-source software distribution (e.g. Linux), very large amounts of data (tens of Gigabytes) must be disseminated to a very large community of developers and users (up to millions of members). Moreover, content is frequently updated to new versions of the software modules. For a Linux distribution, content is generally disseminated either as ISO images of a full Linux release, or as packages that group binaries or source code for a single software module. The problem with the first approach is that successive Linux releases have many common parts that users will uselessly download several times. The finer granularity in the second approach requires more complex data management, with frequent package updates and freshness problems.

The main requirements for an open-source software distribution system could be summarized in four points: (i) avoid excessive charges on the distribution servers and on the communication lines, that lead to poor global performances; (ii) provide advanced search of content based on metadata properties; (iii) provide support for maintaining freshness on content for each user in the distribution network; (iv) ensure robustness in case of failure of some system components. Current distribution

Please use the following format when citing this chapter:

Abiteboul, S., Dar, I., Pop, R., Vasile, G and Vodislav, D., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 209–215.

architectures, centralized or based on a set of mirrors, fail to fulfill these requirements. We believe that peer-to-peer (P2P) architectures, that uniformly share the effort among participants and provide replication, are a good solution for software distribution.

We present here the content distribution solution proposed in the context of the EDOS European project [5]. EDOS stands for *Environment for the development and Distribution of Open Source software* and addresses the production, management and distribution of open source software packages. We only present the EDOS content distribution system, which proposes a P2P dissemination architecture including all the participants to the distribution process: publishers, mirrors and clients. The system was implemented as an application to the distribution of Mandriva Linux packages.

The main contributions of our system are: (i) aP2P architecture providing resource sharing, load balancing and robustness; (ii) advanced information system capabilities, based on distributed indexing of XML content metadata; (iii) efficient dissemination based on clustering of packages and multicast; (iv) support for freshness maintaining on updates, by using subscription/notification. Due to space limitations, the paper only presents an overview of the EDOS distribution system –a detailed description may be found in [8].

2 Related work

Linux distributions use various dissemination methods (an overview is presented in [7]), based on sets/hierarchies of mirrors in most cases, on notification channels in RedHat Network [15], or on versioning repositories in Conary [4].

P2P architectures for content distribution mainly address load balancing and bandwidth sharing (Coral [13], Codeen [3]). We extend this primary use by adding a *distributed information system* based on XML metadata indexing and querying, together with efficient *file sharing and multicast dissemination*, such as BitTorrent [12]. Among the various P2P infrastructures [11], the most appropriate in our context are *structured overlay networks* (Chord, Pastry, CAN, ...), that provide better performances for locating and querying large quantities of data. We use Free Pastry, an implementation of the Pastry [14] distributed hash table system.

3 System functionalities

The goal of the EDOS distribution system is to efficiently disseminate open source software (referred at a more general level as *data or content*) through the Internet. Published by a main server, data is disseminated in the network to other computers (mirrors, end users), that get copies of the published content.

EDOS system is articulated around a distributed, P2P information system that stores and indexes *content metadata*. This metadata-based information system allows querying and locating data in the EDOS network.

The choices for the functional architecture are driven by the three main aspects that define the system: the data model for the content management, the actors and their roles in the P2P architecture, and the usage scenarios.

3.1 Data model

There are *three types* of data units employed by the EDOS distribution system:

– **Package**: the main data unit type, represented by an RPM file;
– **Utility**: individual file used in the installation process;
– **Collection**: it groups together packages, utilities or sub-collections, to form a hierarchical organization of data.

A *release* is a set of data units that form a complete software solution -it corresponds to a full Linux distribution. Its content is described by a collection.

Content dissemination is initiated by *publishing* data units in the system. Publishing consists in generating metadata for each data unit and indexing it in the distributed system. Periodically, the main server publishes a new release. Updates to the current release are realized by publishing new versions of packages or utilities. At some moment, the main server decides to transform the current status of the current release into a new release.

Metadata management is a key issue in the distribution process. We aim at building a global, distributed information system about data to be disseminated in the network. This system is fed with content metadata. The ability to express complex queries over metadata and to provide effective distributed query processing is a major contribution of this project.

In the largest sense, metadata consists in the set of properties that characterize data units. We classify metadata properties in three main categories:

– *identifiers*, i.e. properties that uniquely identify a data unit –in our case, the name and the version number of the data unit.
– *static properties*, that do not change in time for a content unit, e.g. size, category, checksum, license, etc.
– *changing properties*, i.e. properties that may vary in time: locations of replicas in the network and composition for collections.

The XML structure chosen for EDOS metadata is a compromise between efficiency needs for both *query processing* (that requires large XML files, containing all elements addressed in a query) and *metadata updates* (that need small files). Our choice is to create separate XML files for each package (package properties) and for each release (release composition).

3.2 Actors, roles and usage scenarios

Peers of the EDOS P2P distribution system maybe classified in *three categories*:

1. **Publisher**: the main distribution server, that introduces new content in the system. Its roles are to *publish* the new content in the distributed index, to manage client *subscriptions/notification*, and *flash-crowd dissemination* of data, as explained below.

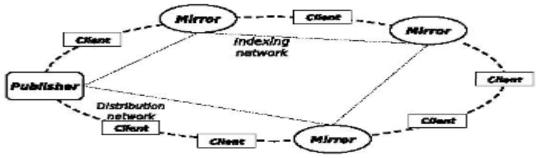


Figure 1 presents the actors in the P2P distribution network.

2. **Mirrors**: secondary servers and trusted peers. Mirrors and end-users have similar roles: *download* content from other peers, query the system and *subscribe* to new data. The main role is to keep copies of the published content, providing additional downloading sources in the network. Unlike end-users, Mirrors are trusted peers, that can participate in *index management*. Also, they are rarely unavailable and provide better QoS.
3. **Clients**: end-user computers, not trusted for index management. They need an entry point to the indexing network for querying the metadata.

Figure 1 presents the actors in the P2P distribution network. Actors are connected in two distinct networks:

– *The distribution network*, composed of all the peers - they store, download and share EDOS data, i.e. software packages, utilities and collections.
– *The indexing network*, composed of trusted peers (Publisher and Mirrors) - they store the index on content metadata. For security reasons, Clients are not allowed to participate in metadata and index sharing, but can provide content, whose validity may be verified by using checksums.

There are two main distribution cases: flash-crowd and off-peak.

Flash-crowd distribution corresponds to situations where new, popular and large size content is published (typically a new release), and many users want to get this content as soon as possible. Flash-crowd distribution uses efficient dissemination methods, based on clustering of data units and multicast. Each user asking for the new release may already have some of its packages -therefore he computes a *wish list* containing only the missing data units. Based on the wish lists gathered from users, the Publisher computes the clusters of data units to be disseminated. Each user will only download the minimal set of clusters that cover its wish list -download is realized in parallel for all the users in a common *multicast* process.

Off-peak distribution corresponds to periods between flash-crowd situations. During these periods, the Publisher may publish updates to the current release, Mirrors and Clients may query the system, download query results, subscribe to distribution channels, receive notifications on such channels and download software updates.

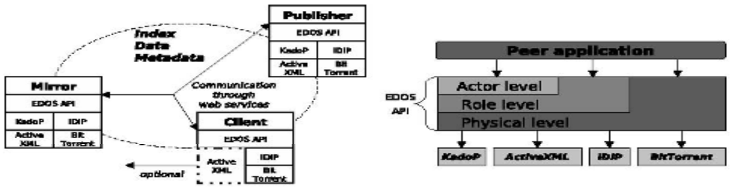


Fig. 2. Software modules and API structure in the EDOS distribution system

4 Architecture and implementation

The EDOS distribution functionalities have been implemented as a Java API, based on a set of external software modules:

- **ActiveXML** [9, 1]: provides an extended XML format for EDOS metadata, storage for metadata documents published in KadoP, and web services for inter-peer communication.
- **KadoP** [10, 6]: distributed index for (Active)XML documents, that allows publishing, indexing and querying EDOS metadata. Based on the Distributed Hash Table (DHT) system Pastry [14], KadoP uses the ActiveXML module as a local repository for metadata documents KadoP decomposes ActiveXML documents into key-value pairs stored in the DHT, and uses this decomposition to compute answers at XML query processing.
- **IDiP**: dissemination platform that implements functionalities for the flash-crowd usage scenario: content clustering and multicast dissemination.
- **BitTorrent** [12, 2]: the well-known filesharing/downloading system, that optimizes the transfer of large files between peers. We use a slightly modified version of *Azureus*, a Java implementation of BitTorrent, for multicast in IDiP and for download from multiple replicas.

The structure of the EDOS distribution API is presented in Figure 2. The API is organized on three levels:

1. **Physical level**: lowest level, provides EDOS peer basic functionalities. The physical level is composed of several modules: a content manager for local content, an index manager for the distributed index, a channel manager for subscription, a dissemination manager for flash-crowd distribution, etc.

Programming distribution applications at the physical level requires more effort, but offers the greatest flexibility.

2. **Role level**: built on top of the physical level, provides a default implementation for each role in the distribution network, i.e. publishing, downloading, replicating, querying, and subscribing.
3. **Actor level**: highest level, provides a default implementation for each actor type (Publisher, Mirror or Client), by combining several roles.

The first **prototype** of the EDOS distribution system is implemented as a set of web applications on top of the EDOS API (seeFigure2). Each peer in the EDOS network runs a Java/JSP web application -there is an application for each actor type: Publisher, Mirror or Client. Peer applications use a Tomcat web server for deployment, with Axis for web services.

The Publisher web application allows publishing new content, managing subscription channels and driving the flash-crowd dissemination process. Mirrors and Clients have the same user interface, allowing queries, downloading, subscriptions to channels and notification handling.

Tests with the first prototype demonstrated the relevance of P2P-based solutions for large-scale content distribution, the ability of managing very large amounts of metadata with KadoP and the improvements brought by IDiP for flash-crowd dissemination. More details are presented in [8].

Next steps will address intensive testing in a real large scale network such asGrid5000 and improvements in massive publication of metadata, in security (peer authentication), in firewall/NAT traversal, in the user interface, etc.

References

1. ActiveXML web page. http://activexml.net.
2. BitTorrentprotocolspecification.http://wiki.theory.org/BitTorrentSpecification.
3. Codeen. http://codeen.cs.princeton.edu.
4. Conary software provisioning system. http://wiki.rpath.com/wiki/Conary.
5. EDOS project: Environment for the development and Distribution of OpenSource software. http://www.edos-project.org.
6. KadoP web page. http://gemo.futurs.inria.fr/projects/KadoP.
7. EDOS deliverable 4.1: Distribution of code and binaries over the Internet, 2005. http://www.edos-project.org/xwiki/bin/view/Main/D4-1/edos-d4.1.pdf.
8. EDOS deliverable 4.2.2: Report on the p2p dissemination system, 2006. http://www.edos-project.org/xwiki/bin/view/Main/D4-2-2/edos-d4.2.2.pdf.
9. S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, and R. Weber. Active XML: Peer-to-Peer Data and Web Services Integration. In VLDB, 2002.
10. S.Abiteboul, I. Manolescu,and N. Preda. Constructing and querying peer-to-peer warehouses of XML resources. In V.T.ChrisBussler, editor, Second International Workshop on Semantic Web and Databases (SWDB). Springer-Verlag, 2004.
11. S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. In ACM Computing Surveys, 2004.

12. B. Cohen. Incentives Build Robustness in BitTorrent. In Proceedings of the Workshop on Economics of Peer-to-Peer Systems, 2003.
13. M. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with coral. In 1st USENIX/ACM Symposium on Networked Systems Design and Implementation, 2004.
A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In IFIP/ACM Middleware, 2001.
14. S. Witty. Best practices for deploying and managing linux with RH Network.

Reusing an open source application – practical experiences with a mobile CRM pilot

Jyrki Akkanen[1], Hunor Demeter[2], Tamás Eppel[3], Zoltán Ivánfi[2], Jukka K. Nurminen[1], Petri Stenman[4]

1 Nokia Research Center, P.O. Box 407, 00045 NOKIA GROUP, Finland
jyrki.akkanen@nokia.com, jukka.k.nurminen@nokia.com

2 Nokia Research Center, P.O.Box 392, H-1461 BUDAPEST, Hungary
hunor.demeter@nokia.com, zoltan.ivanfi@nokia.com

3 Budapest University of Technology and Economics, Budapest Pf. 91, H-1521 Hungary; peletomi@gmail.com

4 Nokia Ventures Organization, P.O. Box 407, 00045 NOKIA GROUP, Finland; petri.stenman@nokia.com

Abstract. We discuss experiences in extending an open source CRM application to develop a new server-based mobile business application. Combining the application code reuse with incremental development process allowed successful development of a pilot application in a tight schedule. In particular, it enabled a quick start for customer-driven development, diminished risks related to the baseline application itself, and provided the flexibility needed in experimental pilot development.

1 Introduction

New software is often developed over a platform. In many senses platforms are ideal for developing new applications: they are usually well-documented and designed for reuse and easy development. However, in this paper we explore a different approach: we discuss the benefits and challenges associated to reusing an existing open source application in order to create a new product. This approach has potential since it allows a quick start and, when combined to an agile process, also an early user involvement. Furthermore, using an open source product as the development base reduces risks due to associated information openness.

Our study grew up from practical experiences. Our vision was to create a business software solution for small and medium-sized companies, a solution that could support both multiple business domains and mobility. To begin with we implemented a pilot system for the Finnish real estate agencies by extending an open source CRM application to a new business domain and by integrating mobile phones in it. The solution was a new concept and included a lot of uncertainty both on the technical and business side. To deal with the uncertainty the venturing literature suggests testing the underlying assumptions at the lowest possible cost (see e.g. [1]). So we were not implementing an application fulfilling a fixed specification but in a very dynamic fashion exploring for the right combination of business and technology to create profitable business.

2. Course of the development project

A lot has been written about motivations to participate in open source development; both from individual and corporate viewpoint [2]. One of the frequently cited benefits is that reusing open source reduces development cost [3,4]. This was also our main motivation: using an open source business application as development base would save development effort and thus allow us to rapidly test our concept and assumptions with a real case.

On the Internet we found roughly 40 open source CRM and group work suites. With a set of criteria (e.g. functionality, quality, license, environment) we finally selected vTiger [5] as our base application. Most solutions were dropped due to lacking functionality or immaturity, and none of them fully satisfied all the criteria. With vTiger we were mostly concerned about the product quality and maturity. It is a web-based business management software written in PHP and running over Apache web server and mySQL database. The product is developed mainly by an Indian company vTiger, which is getting revenue from support and subscription services. Our baseline version was vTiger 4.2.

We used a very light-weight and incremental development process along the common open source guideline "release early, release often" [3]. Our initially rather vague understanding of customer requirements grew during the development, and we also became more and more aware of our abilities to fulfill them. It was only after two months, in the middle of the project, when we finally were able to estimate how much functionality we actually can deliver, and were able to even roughly pin down the final feature set for the deliverable:

1. Basic CRM functionality: contacts, messages, phone calls, notes etc. that could be cross-linked to each other.
2. Real estate business specific functionality: property management and search, bidding process, checklists, brochure creation.
3. Mobility support: synchronize CRM data (e.g. contacts) with phone, rich context to phone calls.
4. PC-phone interoperability: dock the phone to PC, respond to incoming calls on the PC with proper CRM context, and initiate calls by clicking special hypertext link in PC browser.

As depicted in Figure 1, all the business functionality was implemented in an Application Server that provided a web UI for a browser. That consisted mostly of vTiger CRM augmented with Realtor Extensions, the set of specific modules for real estate business. The mobile phones connected to the Application Server through a proprietary Mobility Server that took care of all the mobility-related issues. A small adaptive module, Mobility Connector, in the Application Server provided a Web

Services interface for the Mobility Server. This design made the mobility framework independent from the Application Server.

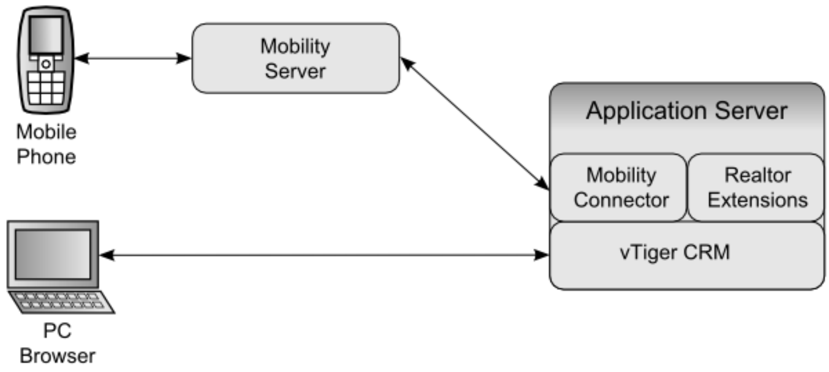


Fig. 1. The solution architecture

We managed to deliver the pilot to our customers according to the original schedule and they were satisfied with the quality of the deliverable. There was no need to fix bugs during the 2 month pilot testing period. The table 1 below sums up the code size in the Application Server divided into different functions.

Table 1. Deliverable size: lines of code including empty lines and comments

	Base	New	Modified	Release
vTiger CRM	361 487	8 000	7 915	373 838
Realtor Extension		12 080		12 080
Mobility Connector		2 979		2 979
Third-party Libraries	163 020	902	380	164 266
Total	524 507	23 961	8 295	553 163

The Base column shows the size of the original vTiger 4.2. The New column contains the lines in the files that we created ourselves while the Modified column tells how many lines in the Base code files were modified. The Release column contains the final size of the deliverable.

The figures show that reuse was very efficient: only 6% of all the code lines in the deliverable were written by us. Furthermore, we used only about 15 man-months to enhance the vTiger CRM into our Application Server, obviously an order of magnitude less than what would have been required for developing everything.

Of course a considerable effort would still have been required to finalize our prototype into a product. In addition, the fact that the vTiger modifications spread over 236 files complicates future maintenance. For instance, merging the changes to a newer version of vTiger would be harder.

Despite the successful pilot we eventually decided to discontinue developing the Application Server. The reasons were mostly related to business: as our revenue would come from mobility support it might be better to rely on existing business applications rather than develop new ones. We thus re-focused our resources in developing the mobility support into such form that creating the necessary Mobility Connector modules for existing business applications would be as easy as possible.

3. Lessons learned

Compared to mature application platforms complete applications have a harder learning curve, especially if they are weakly documented and do not have a clean architecture. Our experience showed that the benefits of complete applications easily outweigh their weaknesses: we got a lot of complete functionality for free. In our case the base application already provided most of the necessary software infrastructure and one third of necessary features. Furthermore, we could develop new functionality on top of the old one with relatively small effort.

Basing our development on an existing application suited well to incremental customer-driven process. Customers easily caught a definitive role in the development process as the lead users could try out working software from the first day of development. We could also quickly provide new functionality: this kept customers active and motivated to provide domain knowledge and feedback. Active weekly customer feedback was crucial to maintain the development speed and right focus. In the software engineer viewpoint, incremental prototyping felt as well the best approach to get familiar with the base application: the source code was readily available, but we did not know it at all and it did not have too much documentation. Making small modifications was a natural way to learn the software piece by piece.

Maybe we could even state that reusing an existing application is an optimal approach only when combined to an incremental process. A careful and deep design phase before implementation requires learning both the existing software and requirements. This is time-consuming and, as a result, one loses the quick start. In worst case one ends up re-designing large parts of the base software in such a way that writing completely new software turns out to be an easier solution.

Incremental prototyping approach is, however, not a silver bullet: it may lead to architectural problems. The architectural decisions in the base software concern also the new features but, in practice, deviations from clean architecture easily occur. In the beginning one simply does not know the software enough to follow its architecture. It is also difficult to control the overall architecture if new features are inserted one after another with least possible effort. In the end some architectural refactoring may need to take place.

A further benefit of open source is that not only the source code, but all other data concerning the software is public and accessible. For us the open information was useful for the software quality. The open source advocates eagerly remind us from the elements in open source development model that promote quality [3] and based on experience with mature top-quality open source products like Linux and Apache, such claims are valid [6]. On the other hand, during our selection process, it became clear that all products were not mature enough for real production use. We encountered security vulnerabilities, weak usability, low performance, lots of dead code, and poor documentation.

Such weaknesses are also common in commercial non-mature software products. However, the benefit of open source development is that it is practically impossible to keep quality problems hidden. To look behind the hype on the front page you can install and try out the software. An experienced developer can recognize low-quality code by looking at the implementation. The discussions in various Internet boards also reveal if users are having problems with the software and indicate what the developer community attitude towards the possible weaknesses is.

The situation is totally different with commercial software. Information can only be accessed "in drops" and getting beyond the front page advertisements often requires commitment. This holds especially if you think about reusing the software in order to further develop it: commercial software is usually not sold or advertised for that purpose.

We were bad citizens in that respect that we did not contribute to an open source community at all. The basic reason for not doing so was that we were unsure about what to contribute, where and how. The realtor-specific features were very domain- and country-specific, so we did not expect them to have much generic interest. The mobility features would surely have been welcomed but the solution depended on proprietary components that could not be open sourced.

An additional difficulty in joining the community was that our own targets were changing as our experience with the business and technology grew. As we were not mature enough to clearly specify our needs we concluded that attempting co-operation in so early stage would just confuse the base product community.

Open source application reuse causes complicated licensing situation inside companies building commercial solutions. When you modify open source code, you end up intermingling your code with the open source code. Depending on the open source license this may lead to many concerns related to commercial usage of the software. You may also end up with problems with your IPR: this is very complicated matter and so difficult to analyze to even understand all the consequences [7]. In our case this led to situation where we had code without clear strategy how to use it later on.

4. Conclusions

The target of our project was quickly and with small resources to provide a pilot of a server-based business application, which supports realtors in their daily work and allows use through a mobile phone. Building on top of an existing open source application allowed us to successfully release the pilot in a tight schedule.

Reusing an open source application gave us a quick start for the development. As the base product already contained lots of functionality, we could concentrate on satisfying the actual needs of the user. Many risks related to the product could be easily managed since open source software came with open information: all relevant knowledge of the base product was easily available.

We also noticed the benefits of combining agile, incremental development process to reusing an existing application. Straight from the beginning of the project the customers were able to use the weekly releases. Their feedback of the features and their usability steered the project and allowed flexible prioritization of the implementation tasks.

On the other hand, for the pilot, we ignored some critical questions concerning the software licensing and our position in the open source community. These must be dealt with before basing a commercial product on an open source application.

References

1. McGrath, R.G and MacMillan, I., *The Entrepreneurial Mindset* (Harvard Business School Press, 2000).

2. Rossi, M., Decoding the 'Free/Open Source Software Puzzle': a survey of theoretical and empirical contributions (2004), in: The Economics of Open Source Software Development, edited by J. Bitzer and P.J.H. Schröder (Elsevier 2005); http://opensource.mit.edu/papers/rossi.pdf.

3. Raymond, E., *The Cathedral & the Bazaar* (O'Reilly, February 2001). See also http://www.catb.org/~esr/writings/cathedral-bazaar/.

4. Bessen, J.E., Open Source Software: Free Provision of Complex Public Goods (July 2005). Available at SSRN: http://ssrn.com/abstract=588763 or DOI: 10.2139/ssrn.588763 (http://dx.doi.org/10.2139/ssrn.588763).

5. vTiger CRM; http://www.vtiger.com/.

6. Boulanger, A. Open-source versus proprietary software: Is one more reliable and secure than the other? *IBM Systems Journal* 44:2 (June 2005), p.239.

7. Vetter, G.R. "Infectious" Open Source Software: Spreading Incentives or Promoting Resistance. Rutgers Law Journal 36:1 (Fall 2004), p.53; http://opensource.mit.edu/papers/vetter2.pdf.

The role of trust in OSS communities – Case Linux Kernel community

Maria Antikainen[1], Timo Aaltonen[2], and Jaani Väisänen[3]
1 VTT Technical Research Centre of Finland , Media and Mobile Usability, Sinitaival 6, 33101Tampere, Finland
maria.antikainen@vtt.fi
2 Tampere University of Technology, Institute of Software Systems, Korkeakoulunkatu 1, 33720 Tampere, Finland
timo.aaltonen@tut.fi
3 Tampere University of Technology, Institute of Business Information Management, Korkeakoulunkatu 8, 33720 Tampere, Finland
jaani.vaisanen@tut.fi

Abstract. Open source software development has been the subject of interest among businesses as well as in the academic world. OSS enables many possibilities for companies but also sets new kinds of challenges. Because of the characteristics of the OSS phenomenon we propose that trust in OSS communities plays a key role in facilitating their success. Therefore, *the aim of this study is to explore the factors that affect trust in OSS communities*. The data is gathered by a survey aimed to Linux Kernel developers. Among other results it may be concluded that the most important factors affecting trust seem to be other developers' skills, reputation as well as the formal and informal practices.

Keywords: OSS community, Linux Kernel, trust, case study, survey

1 Introduction

Open source software development has been the subject of interest among businesses as well as in the academic world. OSS enables many possibilities for companies but also sets new kinds of challenges. In this study we propose that trust in OSS communities plays a key role in facilitating their success. Trust is important in this context because the participants of OSS communities represent different organizations and different motivations, and therefore, there is constant risk for opportunistic behavior between participants. Trust is also a central factor when organizations are making decisions about whether they choose OS software or not.

Therefore, *the aim of this study is to explore the factors that affect trust in OSS communities*. Since the study area is rather unexplored, the study starts with scrutinizing deeply one case that is the Linux Kernel community. The actual study consists of two phases, starting with exploring the trust in OSS communities with the multidisciplinary literature analysis, a phase which is reported in Antikainen and Aaltonen [1]. As a result of the first phase, the preliminary model of trust in the Linux Kernel community was built. In this paper we focus on a web-based survey aimed for the Linux Kernel developers to be able to test our preliminary model of trust.

In this paper we define trust as "the extent to which a person is confident in and willing to act on the basis of, the words, actions, and decisions of another" [2]. Trust occurs in a relationship between a trustor and trust target. Fundamentally, we see that trust implies a party's willingness to accept vulnerability but with an expectation or confidence that it can rely on the other party [3], [4], [5].

2. Linux Kernel community

Nowadays Linux has grown to the largest open source project in the world. [6] Linus Torvalds is still the benevolent dictator of Linux as he is the final arbiter of all changes accepted into the Linux kernel. An often-referred model of an open source community is presented in Nakakoji et al. [7]. The model describes an open source community as an onion, where the most influential roles are in the center, and each outer layer consists of less and less influential ones – see the left-hand side of Figure 1. The right-hand side of the figure depicts the onion-like figure instantiated to the Linux Kernel community.

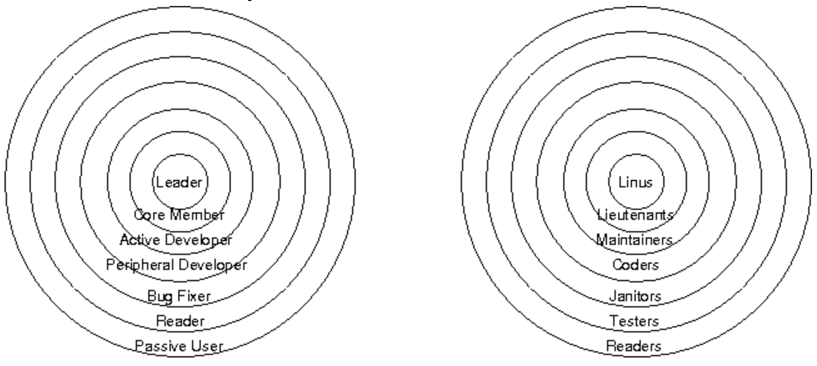


Fig. 1. The general onion model of open source communities vs. the Linux Kernel onion model

Developing open-source software is based on making small changes to an existing code. In Linux a description of such a change is called a *patch*. Certain conventions are followed when a programmer attempts to get her patch "upstream" (to get the patch to the Linus' kernel).[8].

There is no formal process for developing Linux [9]. The unplanned emergence of lieutenants and the convention that patches are delivered to them is de facto behaviour of the community which takes place in mailing lists. Thus, there exists an implicit "control" for organizing the community. People who are skilled and have a large impact in the development simply conceive more power.

Developers who become lieutenants are trusted by the programmers who start submitting their patches to them. Actually there exists a chain of trust in the community, since programmers trust lieutenants who in turn trust Linus. The chains of trust are made explicit by adding signoffs to the patches: the programmer signs the patch she made before delivery, and then if the patch is accepted by maintainer she also signs of the patch and so on. Each patch includes a chain of trust in an explicit form [10]. Precise rules for signing patches can be found at Open Source Development Labs, Inc [11].

3. Methodology

The study uses case study method exploring the Linux Kernel community. The community was chosen based on the structure and the sufficient size of the community, which enabled the collection of the needed amount of data. A web-based survey was created and linked to the Linux Kernel community's main mailing list. The research instrument was administered via a web-based survey tool provider, SurveyMonkey (www.surveymonkey.com). The questions were based on the preliminary framework created in the phase 1. Yet, the questions are not restricted to this framework so that exploratory approach to the phenomenon could be applied.

The study resulted with 139 answers during 12 days. However, since the web questionnaire was divided into several sections, the amount of respondents decreased towards the end of the questionnaire to 95 respondents. Also open questions got fewer responses since they were not obligatory; however, the information from open questions was very valuable.

Analysis of variance was used to gain understanding about the differences between the different factors of trust. Cronbach's Alpha (0,759) was measured to validate the reliability of the questionnaire as well.

4. Factors affecting trust

The survey included questions considering the background of the developers, the nature of their participation in Linux Kernel community as well as questions considering trust. The factors of the preliminary model of trust were asked with multiple questions. In the following the results are briefly discussed and illustrated.

To be able to test the preliminary model and find out the importance of different factors we constructed one common index that is comparable throughout all the factors by calculating the mean results of each factor. The significance of the differences in means was concluded by analysis of variance. There has been discussion regarding the use of ANOVA when dealing with ordinal and grouped variables. Although the formulae behind ANOVA require computations that would normally need scale variables, there is a predominant conception that has been reviewed e.g. in Brockett & Golden [12] that states that the error margin from using interval level methods for ordinal data is very small. Therefore, the authors have decided that the use of ANOVA is acceptable in this situation. The mean values are illustrated in Figure 2.

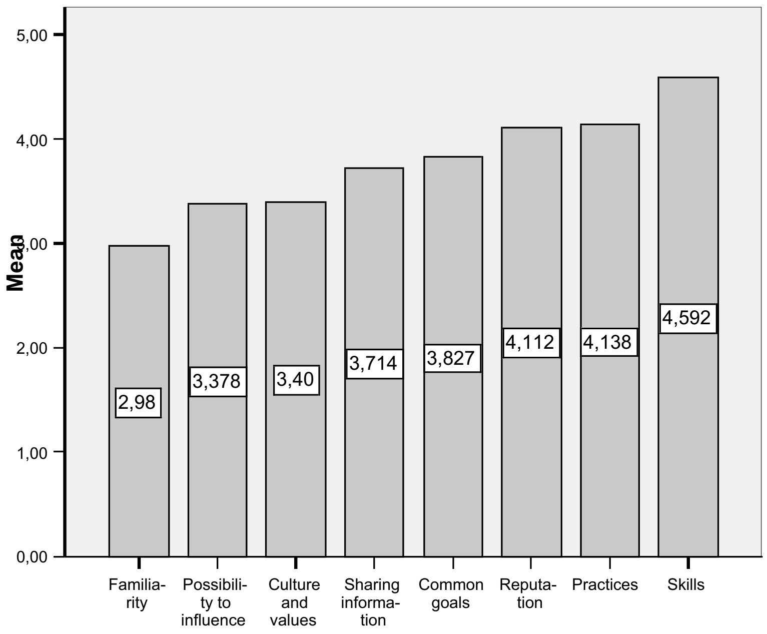


Fig. 2. The means of the factors of trust

Based on the results, 'skills' factor seems to be most important one followed by 'practices' and 'reputation' with almost similar means. Next comes 'common goals' and 'sharing information'. The least important factors seem to be 'the culture and values', 'the possibility to influence' and 'familiarity'. Analysis of variance showed that differences between sharing information and the common goals were not significant. Also differences between 'possibility to influence' and 'culture and values' can derive from coincidence. Again 'reputation' and 'practices' do not have significant differences in their means. Since the values were quite high, based on the analyses conducted, none of the chosen factors can be removed from the model.

The final question considered *how the respondents see the role of trust* in the Linux Kernel community. We got altogether 49 answers to this open question of which 4 were discarded. More than a half of the answers included the statement '*yes'* (26), especially *general yes was* popular. Some of the respondents who answered yes specified their answers adding that they *trusted towards the core of the onion* (see Figure 1), or *towards outer layers of onion. Trust towards the core of the onion* was considered less important, and *trust towards outer layers* was mentioned only couple of answers. In contrast, we got also an amount of plain *no* answers (3).

Other answers considered the trust issue in general level without commenting directly if they trust into Linux Kernel community or not. We divided these answers into three categories: *trust in code* (11), *trust in process* (3), and *trust based on reputation* (4). However, it can be argued that *trust in process* and *trust in code* may seem to belong to the same category. Then one third of the respondents would have chosen this compound category. This open question seems to affirm the results (Fig.2) that skills, practices and reputation have an important role in trust towards Linux Kernel community.

5. Conclusions

In this paper we studied factors influencing trust in the Linux Kernel community. Based on the results we propose that eight factors presented in our model seem to affect trust in the Linux Kernel community. The most significant factors seem to be trust towards other developers' skills, their reputation and informal and formal practices in the community. Also the answers for open question confirmed this conclusion. Due to the fact that Linux Kernel is one of the most complex open source products available, skills and reputation of other members of the community are naturally appreciated. Explanation for the value of practices may be that the conventional software engineering methods are substituted by informal practices. Without them the community would not function.

The least significant factors are familiarity, the possibility to influence and the culture and values. The Linux Kernel community is extremely large, and people are not very familiar with each other. Therefore, familiarity cannot be ranked high. Linux – like most open source projects – is based on a benevolent dictatorship. This fact and the size of the community set limits to possibility to influence. Maybe people willing to gain much influence do not join this kind of communities. Linus Torvalds is a very pragmatic person and he makes sure that Linux is not considered as free software but open source and in such atmosphere the culture and values cannot be significant factors. Therefore, applying this survey to smaller communities might lead to different results in this factor.

This study contributes to a clarification of the role of trust in a one type of OSS community. By testing the preliminary model of trust and measuring the importance of different factors, this study brings valuable knowledge on the dynamics of OSS communities. The present study represents an opening for further studies concerning the role of trust in different kinds on OSS communities aimed for building a generalized model of trust in such communities. As mentioned before, surveying several communities might lead to interesting differences between them.

References

[1] Antikainen, M. and Aaltonen, T. (2007). In Helander, N. & Antikainen, M. (eds.), Essays on OSS practices and sustainability. e-Business Research Center Research Reports 36. Tampere University of Technology & University of Technology.

[2] McAllister, D. 1995. Affect- and cognition-based trust as foundations for interpersonal cooperation in organizations. *Academy of Management Journal*, 38 (1), 24-59.

[3] Lewicki, R., McAllister, D. and Bies, R. (1998). Trust and distrust: New relationships and realities. *The Academy of Management Review*, 23 (3), 438-458.

[4] Moorman, C., Zaltman, G. and Deshpandè, R. (1992). Relationships between providers and users of market research: The dynamics of trust within and between organizations. *Journal of Marketing Research*, 29(3), 314-328.

[5] Morgan R. M. and Hunt S.D. (1994). The commitment-trust theory of relationship marketing. *Journal of Marketing*, 58 (July), 20-38.

[6] The Penguin's Window: Corporate Brands From an Open-Source Perspective. *Journal of the Academy of Marketing Science*, Vol. 34, No. 2, 115-127 (2006).

[7] Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K. and Ye, Y. (2002). Evolution patterns of open–source software systems and communities. *Proceedings of International Workshop on Principles of Software Evolution.*

[8] Site *kernelnewbies.org* on the www.

[9] Linus Torvalds' Benevolent Dictatorship. BusinessWeek magazine August 18 2004.

[10] Andrews J. (2004). Linux: Documenting How Patches Reach The Kernel. URL http://kerneltrap.org/node/3180 on the computing news website *Kerneltrap*.

[11] Open Source Development Labs, Inc. (2005). Developer's Certificate of Origin 1.1. At www page http://www.osdl.org/newsroom/press_releases/2004/2004_05_24_dco.html.

[12] Brockett, P. and Golden, L. (1992). A Comment on “Using Rank Values as an Interval Scale” by Dowling and Midgley. Psychology and Marketing, 9(3), 255-261.

Authenticating from multiple authentication sources in a collaborative platform

Quang Vu DANG[1], Olivier BERGER[1], Christian BAC[1], Benoît HAMET[2]
1 Groupe des Écoles de Télécommunications-Institut National des Télécommunications,
9 rue Charles Fourier 91070 Évry France
{Vu.Dang-Quang,ChristianBac,Olivier.Berger}@int-evry.fr
WWW home page: http://proget.int-ery.fr/projects/PFTCR/
2 PhpGroupWare Project
Benoit.Hamet@laposte.net,
WWW home page: http://www.phpgroupware.org

Abstract. This paper presents a proposal to address the need for multiple authentication sources for users of collaborative work platforms. The proposed approach, developed for the needs of GET and Picolibre, relies on a generic solution that integrate groupware servers in a Shibboleth infrastructure. We have developed adapters for this integration, that we contributed to the phpGroupware project. This document should serve as a basis for discussion in order to validate the level of generality of the proposed approach. We hope that this approach can also help maintainers of other collaboration platforms, who want to integrate a park of deployed platforms with external user identification and authentication services, get a better view of solutions available with Shibboleth.

1 Introduction

The Groupe des Écoles des Télécommunications (GET) is composed of several engineering and business schools together with research centres in Paris (ENST), Brest (ENST Bretagne) and Évry (INT), in France. The research teams are made up of more than 600 full-time research equivalents. The range of the researchers' expertise, from technologies to social sciences, enables the integrated approach so characteristic of GET research and fosters its adaptability to new application sectors and new usages in response to current challenges in the field of Information and Communication.

Starting in order for use as a pedagogical platform, Picolibre [1] is a libre-software system developed at GET, and released under the GNU GPL license. It provides a Web-based collaborative work platform built on top of phpGroupware[1] and other libre software tools. Picolibre provides project hosting facilities for small teams of software developers, which was mainly oriented to teaching and research environments[2].

Picolibre integrates several libre software Web applications, but lacked some features in the current stable version, like a wiki engine. However, we have developed an in-house GET platform, based on Picolibre components, integrating new services like a Wiki engine, or a Webdav server, called ProGET [2]. We have integrated these new features in a new generation PicoLibre that is called PicoForge. This new generation is a more generic and complete solution, and available for all as libre software. Several platforms have been deployed at GET, and developers or researchers may be using services of several such platforms, while working on projects initiated on these different platforms.

At the present time, different accounts may be created on these platforms for the same person. Here comes the need of Single-Sign-On (SSO) facilities between these platforms. GET is in the process of deploying soon a federation of authentication systems and applications, based on Shibboleth, for a better integration of its Information System, which is at the present time distributed among the different schools.

Shibboleth is an infrastructure designed to build an identity federation allowing applications and identity providers to share and exchange attributes concerning user profiles, in order to facilitate user identification and authentication in the realm of a deployed identity federation. We investigated the use of that infrastructure, but we should note that users of our platforms may be either already registered in GET's “company directories”, or external contributors unknown to these directories. Our platforms were not interfaced with our company directories to authenticate its users, and manages its own directory. Here comes the need to develop adapters to integrate Picolibre platforms in the coming Shibboleth federation. But even if company users are recognised by Picolibre, through its use of Shibboleth, we still have to support other users not known of companies directories. Such requirement will also be described in our proposed solution.

The same issues, which are addressed here for Picolibre users and GET, may be found also for other networks of collaborative work platforms, for instance among libre software development communities, for authentication into the various software development platforms they use (Gforge, Trac, etc.).

[1] http://www.phpgroupware.org/
[2] The reader will find more details about PicoLibre at picolibre.org/

2 Shibboleth and SSO Service

Shibboleth[3] is a complete open source platform developed for project "Internet2" [4], aiming at building the federation of identity for education institutions and their partners. It is based on the SAML standard (Security Assertion Markup Language), which defines the assertion of authentication and exchange of attributes of users. It supports the SSO service and the authorization, allowing the definition of access control privileges to Web resources, by using user-associated attributes.

The basic architecture of Shibboleth has three components: Identity Providers (IdP), Service Providers (SP), and a "Where Are You From" service (WAYF). **Identify Providers** (IdP) are responsible of user authentication, and to provide the user attributes for the access control process. **Service Provider** (SP) is managing access to the resources. The Apache web server "plugin" *mod_shib* is a module that allows Web pages access control, based on the attributes values defined in the IdP. The service **"Where Are You From"** (WAYF) helps the user to explicitly choose his/her IdP "of origin", selecting the place where he/she may be known as a source of authentication.

When a user wants to access a SP, it is redirected towards an IdP, or a WAYF service for choosing the IdP. Then the user authenticates to this IdP. After successful authentication, the user is directed back to the requested service, but being authenticated, this time. The SP also requires informations that describes the user, and filters these informations for the authorization process. These informations are sent to the service's Web applications in HTTP headers or cookies.

3 Enhanced authentication in Picolibre

At the present time, users authenticate against the phpGroupware application, which provides the basis for users management in PicoLibre. They log-in when they want to access the "virtual desktop" homepage of PicoLibre, which contains the list of the projects to which they collaborate. User may also authenticate directly to one of the other components integrated in the platform, residing on a same Web server, such as the Sympa[5] mailing list manager or the Twiki[6] Wiki manager in PicoForge version.

[3] http://shibboleth.internet2.edu/
[4] http://www.internet2.edu/
[5] http://www.sympa.org/
[6] http://www.twiki.org/

3.1 Standard authentication Scheme in phpGroupware

In general, phpGroupware handles access-permissions to its applications in an autonomous way, for locally authenticated users, basing itself on a local "account", stored in a "directory" operated for instance by a RDBMS like MySQL, or by a LDAP directory. Access to applications is a three phases process. First, the authentication verifies that the user is the owner of an account. Second, the user's profile is determined. Finally, a work session is created and access to modules is granted.

Depending on the physical implementation of the local accounts directory (MySQL, LDAP, ...), it is possible to share the user's profile with other applications deployed on the same networked environment, providing that necessary administrative policies are adopted, and that custom technical adapters are developed, or configuration decisions are taken. There's, at the moment, no “elegant” SSO service facility, that would allow phpGroupWare to grant, to users already known in other parts of the organisation's information system, a "transparent" access to its applications.

3.2 Shibboleth+Apache as an integrator and SSO service

We need SSO service with other applications deployed throughout our company, outside the platform, to which users will have already authenticated. Shibboleth can come and help solve the needs. Hopefully many Web applications we use, like Sympa, or Twiki are compatible with Shibboleth. Thus, Apache and Shibboleth will be able to act as the integrator of their authentications mechanisms (using a distributed Web service approach). Unfortunately, phpGroupWare's authentication mechanisms come short in such a situation. We need to develop a new identification and authentication adapter for the Apache + Shibboleth combination.

3.3 Mixed environment, and legacy

Our platform may be using a Shibboleth federation once adaptors have been added to all its applications. But, as described above, Shibboleth can't be the exclusive authentication mechanism used, due to the fact that we mix company and external users. We then need a way to “bypass” Shibboleth for some of our users. Another issue has to be solved when Shibboleth is used, the local *mapping*, in the applications, between the users that are recognised in Shibboleth, and the internal reference of the local account in the application. Of course, if Shibboleth is deployed prior to setting-up the platform such a mapping is trivial. But it gets worse if it is deployed on an existing environment with many legacy accounts already existing.

Having considered all the constraints above, we propose to integrate our platforms with Shibboleth using a flexible approach. In particular, we try to facilitate the progressive integration of existing deployed instances, thus diminishing the migration burden for administrators and users. As a result, the design of the new authentication system needs to support the following cases. New users, from the company directories, not yet having a developer account on the platform. These users will be able to create a new account in the platform. Legacy users with an account on the platform : when they have an account in Shibboleth, they will be able to *map* their legacy account to the new Shibboleth identity. Legacy users not registered in shibboleth are still be able to use the platform, as before, “bypassing” the Shibboleth SSO engine. New users external to GET, will still be able to apply for registration, as before.

The situation should be similar for any other authentication mechanism than Shibboleth, to which phpGroupWare would authenticate. We then tried to propose a standard *mapping* mechanism which would not be too specific to Shibboleth.

5 Conclusion

We have described a method for integrating phpGroupWare with Shibboleth to allow the use of SSO mechanisms, while supporting several authentication sources. The integration relies a lot on the use of Apache's authentication modules instead of proceeding with an internal phpGroupWare auth. mechanism. There are very few specifics of Shibboleth in this respect, most of the issues being the same if we use other types of Apache auth. mechanisms.

We introduced several options for configuration and adaptation to other environments, in order to achieve the most generic solution. This solution adapts to several situations, like a fully operational Shibboleth environment, or a deployed platform not in sync with the Shibboleth deployment.

Integration of the new modules developed in phpGroupWare has been implemented with the current 0.9.16.011 version, and avoids modifications of its architecture. We have successfully tested a prototype system on two phpGroupWare and Picolibre platforms, with the Shibboleth infrastructure of GET/INT.

Further tests are needed, but the proof of concept is done. And we have a migration path to an integration of deployed platforms in the IS, while keeping legacy accounts.

The SSO facility obtained will help design networks of collaborative platforms that will offer greater usability and flexibility, for a wider adoption both inside companies or among creative communities on the Internet.

Bibliography

[1] Cousin E., G. Ouvradou, P. Pucci and S. Tardieu, 2002, *PicoLibre a free collaborative platform to improve students skills in software engineering*, in: 2002 IEEE International Conference on Systems, Man and Cybernetics, Vol.1, IEEE, p. 564-568.

[2] Berger O., C. Bac, and B. Hamet, 2006, Integration of Libre Software Applications to Create a Collaborative Work Platform for Researchers at GET, International Journal of Information Technology and Web Engineering 1 (3), 2006.

Learning and the imperative of production in Free/Open Source development

Evangelia Berdou
Media and Communications Department, London School of Economics and Political Science, e.berdou@lse.ac.uk

Abstract. This paper examines the role of learning in structuring access and participation in F/OS communities. In particular it highlights the challenges and barriers to access faced by new developers and the expectations of senior developers regarding the mindsets and capabilities of new contributors. It is argued that learning in F/OS is inextricably connected with the demand for continuous production. The evidence presented is drawn from interviews conducted with inexperienced and experienced contributors from the GNOME and KDE projects. The author challenges the view of learning as an enculturation process and the paper contributes to the understanding of power relations among established and peripheral members in communities of practice.

Learning forms an integral part of the experience of participation in F/OS projects and underlies many aspects of collaboration. Given their limited resources, F/OS communities make significant efforts to lower the barriers to entry for new developers. Nevertheless, new developers face a number of difficulties which are associated with different aspects of development and participation. The paper draws on doctoral research [1] to highlight the challenges inherent in the learning process in F/OS communities from the perspective of new and senior developers. Theoretically, the paper contributes to a better understanding of power relations in communities of practice.

2 Background to the study

This section situates the argument within the context of existing contributions related to learning in F/OS communities and outlines the theoretical and methodological framework for the study

Learning features as one of the main motives for participation in F/OS, and learning practices and processes, such as peer-review, are also often regarded as constitutive elements of the F/OS development model. Studies related to learning in F/OS fall into two broad groups. The first consists of studies that examine the role of tools and the technical characteristics of projects in the learning process. The second group includes studies that focus primarily on issues of socialization and joining. Examples from the first group include Shaikh and Cornford's [2] examination of Version Control or Concurrent Version Tools (VCT or CVS) and Baldwin and Clark's [3] examination of the role of code architecture in organizing and inviting participation. A representative example from the second group is von Krogh et al.'s [4] paper on 'Community, joining and specialization in open source software development' where it is argued that newcomers who eventually received a CVS account adopted specific joining scripts, and patterns of behaviour involving levels and types of activity necessary to become a community member. Similarly to other studies, such as Ducheneaut [5],von Krogh et al. adopt an oversocialized view of learning, predicated on the idea that communities are built upon consensus, shared values and continuity.

Theoretically, the research adopted the CoP perspective and employs Foucault's notion of relational power to address its deficiencies with regard to understanding the power relations between central and peripheral members.

The CoP perspective was developed by Lave and Wenger [6] to account for forms of learning that take place outside the contexts of formal education, such as learning by doing, and learning-on-the job. CoP are formed mainly through the pursuit of a shared enterprise. The theory suggests that as new members, termed legitimate peripheral learners, adopt the ways and practices of the community they move from its periphery to its centre. The approach has been frequently applied in the study of F/OS.

In their critical overview of the way the CoP perspective has been appropriated, Contu and Willmott [7] indicate that most studies conceptualize learning as a process of enculturation into the shared values and norms of CoP and regard CoP as locales of knowledge management. Consequently, according to these two researchers, the more radical elements of Lave and Wenger's original framework, namely the way factors such as access to resources can restrict access to positions of initial peripherality and potential mastery are underdeveloped.

The Foucauldian notion of relational power [8] is useful in providing a framework for the study of power relations in CoP, because it helps draw attention to the complex interdependencies that form between members in the course of their shared pursuit[1]. This is particularly helpful in the context of F/OS which are predominantly voluntary. According to the relational view, power is neither a zero-sum game where different actors compete for resources nor something that is given or exchanged, but is a force that creates complex dependencies and invites a diversity of initiations and reactions on the part of the people involved in them.

The use of relational power has methodological implications, since it focuses the investigation on the concrete practices and tactics employed by CoP members to establish a distinctive system of differentiation between integrated and potential members.

The data presented in this paper are drawn from 25 individual, semi-structured interviews with a wide sample of experienced and new developers involved in the GNOME and KDE projects. The interviews were thematically and discursively analysed.

3 Empirical Findings

The findings are organized according to challenges and learning requirements from the points of view of new and experienced developers.

3.1 Learning and contributing: the newbies' perspective

The stumbling blocks to participation for potential F/OS contributors can be organized into three broad categories. First, there are difficulties associated with the technical aspects and tools of F/OS development, such as the use of CVS. Secondly, there are conceptual difficulties related to understanding the development process and the architecture of the program, how they are set up, how things fit and how they are expected to be put together. Thirdly, there are difficulties related to how newbies situate themselves in the development process, and the selection of tasks that are appropriate to their level of skills.

Before they can reach the point of fiddling with the code, new developers have to learn how to download (check-out), build and install the program's sources. This will allow them to run the latest, in-production, version of the code, a prerequisite for participating in the ongoing development process. Installing a development snapshot is far from straightforward. Once a newbie overcomes this initial hurdle and writes a patch the question arises of how to submit it in the correct format. A set of rules is needed that will allow community members to build the submitted code along with the rest of the resources. Both these processes, checking out code and checking in code, require not only a degree of familiarity with the CVS, but also a conceptual understanding of how "things are put together". Moreover, the incorporation of a patch into the main development tree depends not only on its technical merit, but also on its conformance with the maintainer's view of the appropriate features, and its compliance with the architecture of the module.

Although the documentation that is provided often gives information on some of these issues, such as how to use the CVS, it frequently fails to provide answers to the more conceptual aspects of development. Moreover, even if it exists and is updated, finding the appropriate documentation is often an arduous task. One of the problems most frequently indicated by interviewees concerned the fragmented character of the documentation and other online resources. Neal (19/10/2004), another newbie contributor, indicated that often the information required was not offered in the form of a dedicated resource, but was obtainable from developers' blogs. According to Neal, blog syndication sites, such as Planet GNOME and Planet KDE, are useful because they centralize development information, despite the fact that developers' entries need to be scanned to separate the social, from the technical aspects of information.

This intertwining of the social with the technical, highlighted by Neal, which is characteristic of CoP, may account for the frequently employed practice of lurking on the project's development mailing lists. The public, archival character of the mailing list and its use as a repository of knowledge makes the posting of a message a non-trivial affair, especially for new developers. As the next section will show, experienced developers usually assign considerable importance to how newbies comport themselves on mailing lists, especially the ones that carry the most important development traffic.

Most of the interviewees considered finding a task that is appropriate for their level of skills as one of the major hurdles to participation. Several interviewees had become involved in the development by first assuming more peripheral tasks, such as translating, or by concentrating on fairly self-contained development tasks such as bug triaging[2]. In certain cases, having tasks or projects explicitly addressed to new developers appears to greatly facilitate participation, not least because their initiators will often assume the role of mentor. Although responsiveness is often seen as being one of the key characteristics of F/OS, getting the right people to pay attention to suggestions and look at work is not straightforward.

Despite, and in some cases because of, these difficulties, many interviewees described their experience of participation and collaboration in F/OS as educational and much more valuable than the formal training that is provided by most computer science degrees. In fact, in many instances, participation in F/OS was framed in terms of vocational training. To summarize, therefore, newbies describe integration as a slow learning process during which they build up their skill sets and their community knowledge and position themselves in the development through their choice of tasks.

3.2 Learning and production: the senior developers' perspective

The discourse of more experienced developers relating to new contributors is informed by a production-oriented view of the development process. This shapes their expectations in terms of the behaviour and performance of newbies and guides their decisions about helping them.

One of the characteristics most valued in new contributors and F/OS developers in general, is self-reliance. Many interviewees stated that new developers often expect to 'have their hands held' and to be assisted every step along the way. The rapid release rate of F/OS development and the fact that many developers work on a voluntary basis makes time a very valuable resource in F/OS. Every minute spent helping a newbie is a minute less on writing code. Combined with the high turn-over rate of contributors is the phenomenon of programmers who say that they want to help but who then disappear, which means that senior developers generally take great care about choosing who to help. One of the first things senior take note of in assessing the potential of new contributors is their chosen entry point in the development and the way they initially present themselves on the mailing lists. Newbies who demonstrate that they have tried to develop their understanding of the project's architecture and have an idea about that tasks that they might be able to

[2] Triaging is a Quality Assurance process that involves confirming good and reproducible bug reports from the projects' bug tracking tools, in order to identify exactly which actions generate faults in the program.

perform are regarded more positively than those who simply ask for general help and guidance.

Senior developers' rules of thumb for assessing the potential of new contributors indicates how very much intertwined are the values of self-reliance, commitment and productivity. Putting in the time and the effort to find things for oneself is an indication of commitment and, at the same time, a prerequisite for sustained participation. Successful information seekers and dedicated learners do not impose on the time and attention of senior developers and the incremental self-relying development of their knowledge, a common characteristic of experienced developers, attests to their potential as productive contributors. Furthermore, seasoned developers usually judge the potential of new contributors very quickly, sometimes even from their first few postings. The way potential contributors introduce themselves to the community is not just a matter of successful 'face-work', a sign of whether or not they have successfully assimilated the behavioural 'scripts' of F/OS development. A newbie's initial postings seem to indicate the extent to which they have already committed themselves to the development process.

4 Discussion & Conclusions

The research indicates that although the importance of helping out new developers is generally recognized and attempts to organize and facilitate their integration are reflected in the existence of mailing lists specifically set up for this purpose and the provision of tutorials and documentation, it is also understood that peripheral participation is something that needs to take place in the background and not at the forefront of development. New developers are generally expected to orient themselves by making do with whatever learning resources are available and gaining a working understanding of the project before seeking the help of experienced developers. New developers who seek help on specific issues having demonstrated an active engagement with the project are generally considered more promising than newbies who ask for general help and guidance. The investigation of the dynamics of cooperation between senior and new developers suggests that the role of learning in F/OS communities goes beyond that of establishing a common framework of shared values, practices and networks of contacts between peripheral and central members. The analysis of the interviews indicates that learning processes are integral to the exercise of power and control. The significant barriers to entry, are viewed by senior developers as necessary elements of a process that ensures the level of commitment and capabilities required of new contributors. These barriers indicate that even access to positions of initial peripherality is structured.

A possible explanation for these two seemingly inconsistent strategies, community efforts on one hand to lower the barriers to participation and experienced developers' strategies for attracting the 'right type' of contributors on the other, can perhaps be found in the inherent tension that exists between the need to attract and integrate capable volunteers and the demands of continuous production. F/OS are not simply communities set up as knowledge management locales; they are primarily communities organized around the production of a complex good, software. A significant differentiating factor compared to traditional apprenticeship contexts concerns the minimum degree of commitment demanded in order to be recognized as a legitimate peripheral learner. In F/OS individuals can contribute as much as they want, to any level they went and when they want. However, the ease of signing up combined with the appeal of being known as a F/OS developer means that there will always be more potential candidates, as indicated by the high degree of turn-over of contributors, than legitimate peripheral members. As a consequence, it seems that the criteria for being recognized as a potentially valuable contributor in F/OS differ substantially from those for offline professional networks and communities of practice where institutional frameworks, formal employment relations, formal accreditation schemes and tighter social networks ensure a certain level of skill and some degree of continuity and commitment.

This paper provides a basis for understanding the role of learning in structuring access and participation in F/OS communities. One of its major limitations is that it does not examine failed cases of legitimate peripheral participation, but focuses only on successful ones. In addition, the study did not take account of the contextual factors of learning and participation, i.e. how the issues of culture, language and the existence of a supportive network might affect legitimate peripheral participation.

The research was funded by the Greek State Scholarships Foundation (IKY). In its later stages the study was supported by the EU funded OPAALS project (contract no: 034824). The views expressed in this paper are those of the author.

References

1. Berdou, E., Managing the Bazaar: commercialization and peripheral participation in mature, community-led Free/Open Source Projects, in Media and Communications Department. Forthcoming 2007, London School of Economics and Political Science: London.
2. Shaikh, M. and T. Cornford, Version Control Tools: a Collaborative vehicle for learning in F/OS. 6th International Conference on Software Engineering proceedings - Collaboration, Conflict and Control: The 4th Workshop on Open Source Software Engineering, Edinburgh, Scotland, May 25th, 2004.
3. Baldwin, C. and K. Clark, The Architecture of Participation: Does Code Architecture Mitigates Free Riding in the Open Source Development Model? Harvard Business School Working Paper, Final Version, 2005.
4. von Krogh, G., S. Spaeth, and K.R. Lakhami, Community, Joining and Specialization in Open Source Software Innovation: a Case Study. Research Policy, 2003. **32**: pp. 1217-1241.
5. Ducheneaut, N., Socialization in an Open Source Software Community: a Socio-technical Analysis. Computer Supported Cooperative Work, 2005. **14**(4): pp. 323-368.
6. Wenger, E. and J. Lave, Situated learning: Legitimate Peripheral Participation. 1991, Cambridge: Cambridge University Press.
7. Contu, A. and H. Willmott, Re-embedding Situatedness: The Importance of Power Relations in Learning Theory. Organization Science, 2003. **14**(3): pp. 283-296.
8. Foucault, M., The Subject and Power, in Michel Foucault: Power/ Essential works of Foucault 1954-1984. Vol. 3, J.D. Faubion, Editor. 1982, Penguin Books: London; New York. pp. 326-348.

Open source technologies for visually impaired people

Patrizia Boccacci[1], Veronica Carrega[1], and Gabriella Dodero[2]
1 DISI, University of Genova, Via Dodecaneso 35, Genova, Italy
boccacci@disi.unige.it
2 Free University of Bozen, Dominikanerplatz 3, Bozen, Italy
gabriella.dodero@unibz.it

Abstract. We describe two open source applications which we have experienced as very useful aids for the integration of people suffering from visual impairments, from hypovision to actual blindness. The first application is based on speech synthesis and has been experienced by disabled university students. The second experience is oriented to schoolchildren with low residual vision, and it provides their educators and parents with easy to use tools for image manipulation, especially designed for exploiting residual visual abilities.

1 Introduction

The increasing interest in the promotion of education for visually impaired students, at any level, has correspondingly increased also the number of blind university students, who successfully graduate in all subjects. At the same time, EU and Italian legislation are enforcing the adoption of accessible technologies to help disabled citizen in their studies, work and social life. In the same countries there have been also many actions in support of open source technology adoption within public administrations. Yet today most software to be used as computer based aid for disabled citizens is proprietary, and it is often acquired with public funds, by schools, universities and by National Health Services.

The possibility to use open source aids for disabled people has been investigated by the authors, having in mind the specific needs of a blind student, who enrolled in Fall 2005 as BSc student in Physics. Desktop Linux distributions, which already integrate speech synthesis tools, could not be installed in student labs because data acquisition hardware, used for performing physics experiments, was not supported under such distributions. As a consequence, a thorough study of what OS speech synthesis software was available was undertaken, and finally a lab workstation was configured, running open source software, and software available for free to non-profit organizations[1]. This workstation provides speech synthesis in Italian and in English under Linux, with easy keyboard commands to switch between the two languages. No Braille device needed to be installed, only earphones. While speech synthesis may substitute reading, information conveyed by images cannot be perceived by blind students. Students with low residual vision may still view images by means of optical devices (magnifiers) enlarging them, but such tools are heavy and expensive, and a student can seldom use them both at school and at home. For this reason, we investigated the features of open source image manipulation tools, and developed an easy to use interface, especially conceived for the educators and parents of visually impaired children. The image manipulation tool performs contour extraction and "thickens" the lines so that images can be clearly perceived. In the rest of the paper we shall describe how we use open source speech synthesis tools, and then the image processing tools which we have developed. We conclude with some comments on this experience, and on its relevance for the open source community.

2 Speech synthesis

Speech synthesis is the process of generating a human voice by means of electronic devices connected to a computer. It consists on two main steps. The first step is text analysis, where the input text is transcribed in a phonetic representation. The second step is the generation of sounds (waves), where audible output is made with a combination of phonetic and prosody information. The process is sketched in Figure 1. The software module implementing the first phase is often called NLP (Natural Language Processor), while the second is called DSP (Digital Signal Processor).

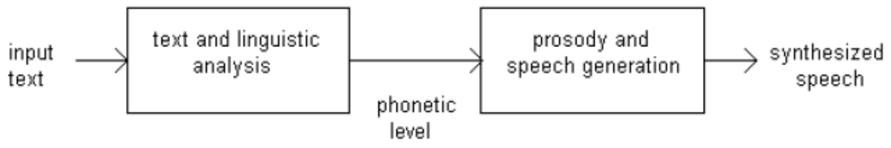


Fig. 1 . Organization of a speech synthesis suite

NLP contains translations from letters to sound, and prosody generation, but also a morphological and syntax analyzer. Extraction of a syntax tree from the sentence is in fact extremely useful to achieve good phonetical and prosody translations.
Most DSPs use a voice database, that is a collection of registered speech fragments. Elementary components of such database are sounds (a vowel or a consonant) or better, diphones, which include the second part of a sound and the first part of the next sound (for example the transition between a vowel and a consonant). Starting from phonemes and prosody, the DSP outputs a sequence of segments, which are adapted to the required prosody, eliminating discontinuities; finally the resulting flow is synthesized and the voice is output[2]. Several software tools are available under Linux, but most of them were of no use for us, since they support only English, or provide marginal support to other languages. The main cause is the length and cost of creating a database of sounds containing all the relevant diphones for each language, and the difficulty in expressing and tuning the rules for transforming texts into phonemes.

One of the most promising speech synthesis systems is MBROLA[3], which has a database of phonemes for about thirty languages, including Italian[4].

Unfortunately, the licensing policy of such project is quite restrictive: MBROLA can be distributed only as a precompiled binary, and only non-military and non-commercial applications may use its databases of phonemes. In other words, it is not an open source product, even if it can be available for free, and several open source tools are compatible with it. The BLINUX website[5] provides many (but not all) open source resources for visually impaired users. It collects a huge amount of packages of several kinds, from Braille device drivers to more complex systems like speech synthesis modules. The selection of suitable packages within BLINUX is not straightforward, and of course, care must be taken in order to avoid old systems, no longer compatible with recent kernel versions, or without any more support groups of active developers and users. For speech synthesis, this problem is especially important since there is no single tool solving the problem, and we need a suite of interoperable applications. The next subsections describe our choices.

2.1 Festival

Festival[6] has been developed at Edinburgh University, as a research tool in speech synthesis. It includes a complete text-to-speech conversion module, which can be invoked by a shell, by a command interpreter, by Java or by an EMACS interface. The system is written in C++ and uses the Edinburgh Speech Tools Library for low level operations. It can be used as a standalone tool, or as a development environment for further speech synthesis tools. Both Festival and the Speech Tools are distributed without licensing restrictions, for both commercial and non commercial use. Festival supports several languages, including Italian, by installing additional files. Italian phonetics has been developed at the Padua Laboratories of the National Research Council; such database can be used together with the MBROLA speech synthesizer (as we did), and provides different voices for the language, including both male and female voices.

2.2 Speech Dispatcher

Since the lab will be used in various scientific related tasks, it is important to integrate speech synthesis within an environment like EMACS. A system capable of connecting Festival to EMACS is Speechd-el and Speechd. The core of the system is Speech Dispatcher [7] (called also Speechd), which provides a device independent level for speech synthesis, across a stable and well documented interface. Speech Dispatcher is implemented as a client-server system: applications invoking speech synthesis functions are clients, which establish a TCP connection to Speechd through SSIP (Speech Synthesis Independent Protocol). Libraries for interfacing to many languages are available (C, Python, Elisp, Common Lisp). Speech Dispatcher translates requests from client applications into commands to the synthesizer, handles message priority, keeps trace of requests logs, and provides the so-called sound icons functionality. Sound icons are sound sequences informing the blind user about the execution of particular actions, such as opening, saving and closing a file.

Once a client application has established a connection, it invokes functions like say( ), stop( ), pause( ). Speech Dispatcher parses such commands, reads the text to be pronounced, and queues it in accordance with requests priority. Then it selects when, by which synthesizer, and with which parameters such a text shall be output. In this way, applications need not deal with low level details in speech synthesis.

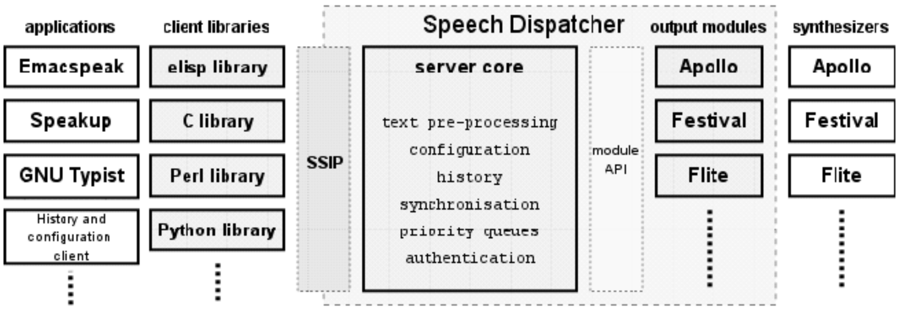


Fig. 2. Speech Dispatcher architecture and interfacing

Figure 2 shows the structure and links between Speech Dispatcher and applications on one hand, speech synthesis systems on the other hand. In fact, Speech Dispatcher interfaces with many speech synthesizers, including Festival. Speech Dispatcher interfaces with EMACS by means of the libraries Festival-freebsoft-utils, and through the Speechd-el client. The latter is a voice output system for EMACS which provides an Emacs Lisp library in order to make available a subset of Speech Dispatcher functions. This makes EMACS totally accessible to blind users.

2.3 Gnopernicus

Gnopernicus[8] is a system especially designed for hypovision; it allows people with low residual vision, and even blind Braille readers, to use the GNOME desktop and its applications. It provides Braille output, screen magnification, screen reading; it may interface to synthesizers like Festival. It provides an Assistive Technology Support menu, which can be used to configure system startup and activate Gnopernicus at login, making the blind user totally free from external help.

3 Image Processing

By discussing with therapists, educators and parents of visually impaired children, we realized that simple magnification of an image contained in a textbook may not be sufficient to provide relevant information to disabled pupils. In fact, images often contain details, irrelevant to the intended educational purpose. Such details are automatically discarded by normal children, while for visually impaired children their selection wastes much effort. Also the use of many bright colors, intended to capture the attention of normal children, can be confusing for a disabled pupil, who would most appreciate a line drawing with contrasting background and no further details. The most severely impaired people, who cannot perceive an image with color or greyscale details, often may see such a simple sketch drawn with a thick line. There exist already several open source tools, which may perform suitable image processing and may make digital images better perceivable to hyposeeing people. However, none of such tools has been designed for being used in such a specific domain, and the desired functionalities should be sought for within menus and options, discouraging parents and educators from using them. To help non specialists to convert, automatically, color or greyscale digital images into sketches visible by hyposeeing people, we developed a plugin which extends the features of a well known open source tool, ImageJ. ImageJ is a public domain Java image processing program that runs, either as an online applet or as a downloadable application[9]. ImageJ has an open architecture that provides extensibility via Java plugins. Custom acquisition, analysis and processing plugins can be developed using an editor and Java compiler. Our plugin provides a simplified interface which skips all ImageJ menus containing unnecessary features. The user, through an accessible menu, can set her/his personal profile, selecting between three degrees of resolution (low, medium,high), three thicknesses of line (thick, medium, fine) and color pairs for background and lines (black/white, yellow/blue etc). The plugin converts color images into grayscale, and then applies a standard algorithm[10] of edge detection, using threshold computed from user profile data. Figure 3 shows an original image, and the resulting image after conversion with our plugin: only essential details are kept, and the resulting image is visible to seriously hyposeeing people.

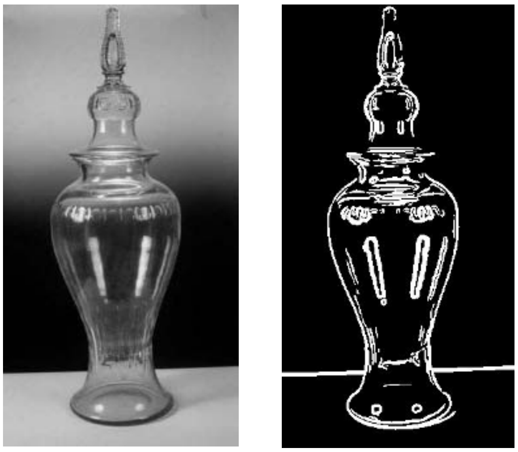


Fig. 3. Original and converted image of a glass bottle

4. Conclusions

Aids for disabled people represent an application domain which seems especially suited for the open source paradigm, since the unavoidable need for adaptation to actual degree of disability is best satisfied by open source software. In practice, this has seldom occurred till now, probably for lack of awareness about the existence of viable open source disability aids. Our experience has shown that existing open source tools may represent good disability aids. Of course, proprietary software aids exist as well, with comparable or higher quality. However, the quality of open source solutions is already sufficient for many users. Someone objected to using MBROLA as a component of our speech synthesis solution, since it cannot be considered open source software. This however may be a temporary solution, since an open source alternative, SMS[11], is almost ready for being released. Use of such software together with Festival shall allow us to rely on open source software only.

The fact that speech synthesis open source software exists does not mean that it is trivial to run it and that our installation shall represent a solution for all hyposeeing problems. From users perspective, knowledge of Braille, presence of residual vision and personal preferences may suggest a completely different approach with respect to ours. And considering Linux installations, care must be taken to select audio peripherals that are already supported by recent drivers, under the preferred distribution. Similar considerations hold for image processing tools: existing software has to be made available through an accessible and simplified interface, in order to be really usable for our purposes. In both cases, we experienced the interaction with developers and users communities, who gave useful feedbacks on how to solve configuration and interoperability problems, and how to modify or extend the available code, whenever needed.

Acknowledgement
The authors are indebted to Silvia Dini from Istituto Chiossone, Genova, for fruitful discussions and feedbacks about possible aids for visually impaired people.

References

[1] V.Carrega, Open Source tools for visually impaired people, Master Theis, Università di Genova (2006), in Italian.

[2] F. Felletti, Text-to-speech technologies applied to website navigation, Master Thesis, Università di Ferrara (2004), in Italian.

[3] T.Dutoit, V.Pagel, N.Pierret, F.Bataille, O.Van Der Vrecken The MBROLA Project: Towards a Set of High-Quality Speech Synthesizers Free of Use for Non-Commercial Purposes, in: Proc. ICSLP'96, Philadelphia, vol. 3 (1996)pp. 1393-1396.

[4] P.Cosi, R.Gretter and F.Tesser, Festival speaks Italian, in: Proceedings of GFS2000, Padova (2000), in Italian.

[5] Blind + Linux = BLINUX, (February 23, 2007) http://leb.net/blinux/

[6] R.A.J. Clark, K. Richmond, and S. King, Festival 2 - build your own general purpose unit selection speech synthesiser, in: Proc. 5th ISCA workshop on speech synthesis (2004).

[7] Speech Dispatcher, (February 23, 2007) http://www.freebsoft.org/ speechd

[8] Introducing Gnopernicus, (February 23, 2007) http://developer.gnome.org/ projects/gap/AT/Gnopernicus/

[9] ImageJ, Image Processing and Analysis in Java, (February 23, 2007) http://rsb.info.nih.gov/ij/

[10] J. Canny, A Computational Approach to Edge Detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 6 (1986) pp. 679-698.

[11] G.Sommavilla, C.Drioli, P.Cosi, G.Tisato, SMS-FESTIVAL A new framework for open source speech synthesis, Proc. AISV conference, Trento, (2006).

Different Bug Fixing Regimes? A Preliminary Case for Superbugs *

Jean-Michel Dalle[1] and Matthijs den Besten[2]

[1] Université Pierre et Marie Curie, Paris, France; jean-michel.dalle@upmc.fr
[2] University of Oxford, Oxford, UK; matthijs.denbesten@oerc.ox.ac.uk

Abstract. The paper investigates the processes by which bugs are fixed in open-source software projects. Focusing on Mozilla and combining data from both its bug tracker (Bugzilla) and from its CVS, we suggest that: a) Some bugs resist beyond the first patch applied to the main branch of the source code in relation to them, which we denote as superbugs; b) There might exist different bug fixing regimes; c) priority and severity flags as defined in bug repositories are not optimized for superbugs and might lead to a involuntary side effects; d) The survival time of superbugs is influenced by the nature of the discussions within Bugzilla, by bug dependencies and by the provision of contextual elements.

There have always been claims according to which open-source software would be structurally able to implement more efficient development methodologies along several dimensions, and notably vis-à-vis reliability. "Given many eyeballs, all bugs are shallow" [4]. In other words, the visibility and accessibility of the source code, in addition to extended peer review processes associated with the existence of a community, would be key to a superior reliability.

However, it is probably honest to say that these claims have never received a true empirical validation, although most data that relate to open-source development is archived online and thus freely available to both open-source projects and academia. Furthermore, a recent study by Coverity[2], a Stanford spin-off company whose technologies allow for the automatic analysis of source code to identify some of its defects, has stressed that mission- and safety-critical proprietary applications were able to reach reliability levels above the most reliable open-source projects. Beyond the straightforwardness of this finding, it still contributes to upgrade the general view on the reliability levels that different open-source projects were able to reach, while quality and reliability are becoming a major concern for open-source software.

Are there indeed general elements in the open-source development methodology that would allow for an intrinsic superiority of open-source in terms of

* The work presented here has benefited from discussions with Patrick Brézillon, Paul A. David, Laurent Daudet, Fabrice Galia, Hela Masmoudi and others. The support of Calibre, an EU FP6 project, and of NSF project NOSTRA in the early phases of this research are also gratefully acknowledged.

[2] http://scan.coverity.com/

reliability and defect density? There is a strong case here for empirical studies, in a more general context of inquiries about open-source software development based on mining available online archives. Indeed, bug fixing processes in open-source software have already received some attention [1, 2, 3, 5]. Following this literature, we further suggest that more could be learnt by mining data from *both* bug trackers *and* code repositories, i.e. the interactions between what happens in Bugzilla and what happens in the CVS. We suggest that there might exists several different fixing *regimes*, and that, among the bugs that tend to resists beyond the first attempts to fix them, there exist superbugs — as we propose to denote them, in a direct analogy with antibiotic-resistant bacterias — for the fixing of which the exchange of context elements between users and developers might be key.

Data We focus here on Mozilla, not only as a prominent example of a successful open-source software project that has already been the subject of various empirical research investigations, but also because it is in the context of Mozilla that the well-know bug repository Bugzilla originates. We combine Mozilla CVS data with Bugzilla data. Namely, for each bug number in Bugzilla, we look for this number in commits to the main branch of Mozilla's CVS, using a heuristic script that we have developed to this purpose. We then combine, for each bug number, some of its characteristics inferred from Bugzilla with other characteristics inferred from retrieved CVS data.

We limit ourselves to relatively old data to avoid censoring biases: i.e. we suppose that a sufficiently long time has elapsed so that we can neglect bugs that would not have been fixed yet. We also did some preliminary cleaning up of the database, removing the first 1000 smaller bug numbers, so as to avoid transitory initial conditions and to control the fact that smaller number have a higher probability to be found in the CVS using our script while not corresponding to bug numbers. As a result of preliminary univariate analysis, we removed some outliers, and typically removed from our sample all bugs that would correspond to either more than 1000 files fixed, that would depend from more than 25 other bugs, and that would be associated with a bug report open in Bugzilla for more than 1500 days before the first patch associated with the corresponding bug number is committed to the CVS. Furthermore, we removed from most of the analysis presented in this paper all bugs whose severity had been set to "enhancement" as they would rather correspond to feature requests properly speaking. Finally, we limited ourselves to all bugs whose resolution had been set as fixed, compared to others resolution types. Ultimately, our database includes approximately 17000 bugs.

Superbugs First of all, many bugs tend to "live" for a long time after the code is first patched in relation to them. There are for instance 650 bugs in our sample that were patched between 10 and 100 days after the first patch was applied in relation to them to the main branch of the code base. Compared to previous studies, these bugs are not simply associated with long discussions in bug tracking system, but also with patches associated with them appearing for a long period of time in the code main branch. Although a considerable fraction of the bugs are corrected on the first day in which the code of the main branch is patched in reference to them, this “long tail” effect is however important. That is to say, some bugs seem to be resistant to the “treatments” that they initially receive: in that sense, we suggest to call them *superbugs*, in an analogy with other resistant life forms that are now developing in hospitals. Compared to other unusual software bugs such as Heisenbugs, superbugs belong to a more general category that might include some of these more peculiar ones.

Moreover, there seem to exist different *fixing regimes*: The hazard function that fits to our data (details upon request) has a ‘bathtub’ shape. A shape well-know in engineering and generally associated with the existence of 3 different regimes: an initial phase, a flat middle one, and a last so-called ‘wear out’ one — in our case below 10 days, between 10 and 100 days, and above 100 days. Within the latter category, it might be that there would even exist another regime above 1000 days. It is absolutely clear to us that this categorization is very tentative, being based only on one open-source project, and we very much hope that future investigations will refine it. However, as a first step to progress in the exploration of bug fixing regimes, we suggest to denote bugs fixed in 10 days or less after a first patch has been applied in the main branch in relation to them simply as *resistant* bugs, while bugs fixed in more than 10 days could be characterized as *superbugs*. And since the latter category could itself include different regimes, we will denote as *hyberbugs* bugs fixed in more than 100 days and 1000 days or less, leaving bugs fixed above 1000 days for future investigations.

Fixing Time: severity and priority flags What factors affect the fixing time of bugs in all three regimes? Obvious candidates are the two variables that are set by bug reporters and developers, respectively, to characterize bugs, namely, severity and priority. severity is set by bug reporters under explicit guidance not to use the blocker and critical levels (severity = 6 and 5, respectively) out of purpose, while the variable priority is set by developers. Our analysis of our data (available upon request) shows that there is no distinguishable pattern, except for severity = 1 (trivial) and 2 (trivial). trivial would seem to result globally in a relatively lower survival probability while minor results in a relatively higher one. These results are confirmed when plotting similar graphs for each of the resistant bug, superbug, and hyperbug regimes: however, more precisely, the minor effect is more apparent for superbugs and hyperbugs, and the trivial effect for resistant bugs. One hypothesis here is that, due also a limited number of bugs associated with severity 1 and 2 that tends to show that these two categories aren’t used very often by developers, trivial bugs might be eliminated more rapidly precisely because of their triviality i.e. *easiness to correct*, thus in the resistant bug regime, whereas bugs flagged as minor would on the contrary could tend to be *neglected* compared to others of higher severity, an effect that would naturally be more pronounced as times goes on, i.e. for superbugs and hyperbugs. If so, it might be worth simplifying the number of severity categories in bugzilla by typically avoiding minor and flagging bugs as trivial or normal or higher. Furthermore, the only distinguishable pattern for priority is for priority = 5 (P5): bugs flagged at a very low priority tend to be patched earlier. This surprising finding is probably to be related to a different use of priority flags under different regimes. There are no P5 bugs among hyperbugs. An explanation for this would be that working on bugs of low priority is abandoned: not that these bugs are necessarily fixed, but they do would not survive for *lack of interest*. The fact that the priority variable is set by developers, compared to the severity variable by bug reporters, would tend to support this explanation.

Fixing Time: survival analysis It is possible to fit predictive models of the fixing time of bugs in all 3 regimes. The linearity of the survival function in loglog vs. log scale suggest that using a Weibull distribution approximation is reasonable as a first step, although the actual distribution is most probably of a power-law or similar nature at least for resistant bugs and superbugs Table 1 synthesizes results of survival analysis regressions (detailed results available on request).

First, both priority and severity appear effective for resistant bugs, and less so for superbugs and hyperbugs. Some of the priority levels influence survival time counter-intuitively on the full sample, which we again interpret as resistant bugs and superbugs with low priorities being abandoned. The highest two levels of priority (and when priority is missing) tend to reduce the survival time of superbugs compared to P5. With due cautiousness due to the improper fit of the model, two levels of severity, minor and major, seem to have an influence in increasing and decreasing, respectively, the survival time of hyperbugs compared to blocker level. The minor effect, already presented above, is indeed also present on the full sample. The interpretation of the negative major effect on hyperbugs and of the negative critical effect on the entire sample are less clear. More interestingly perhaps, different levels of severity influence the survival time of bugs differently: normal has a less pronounced effect than critical, major and minor, compared again to blocker. Two different effects might be at play here: a minor effect, again, associated with neglect, and maybe a difficulty effect, critical and major being just more difficult to fix or implying more cautiousness, discussions and care.

Second, both resistant bugs and superbugs are affected by dependencies, again counter-intuitively: bugs that block many other bugs appear to be fixed *less* rapidly, while bugs that depend upon many others are fixed *more* rapidly. We interpret the first part of this finding as a probable consequence of the difficulty of fixing bugs that block many others: that is, the fact that a bug blocks several others indirectly is an evidence of interdependencies that render its fixing lengthier. The second part of this finding is less clear. It might be, since *bug report networks* play an important role in bug fixing processes as has been recently suggested [5], that bugs inserted in bug report networks would tend to attract more *attention* from developers: and dependent bugs would be fixed relatively rapidly once the bugs that blocked them would have been fixed,

Table 1. Significance and impact of variables controlling for bug fixing regimes. The source of the data is indicated with B for Bugzilla and C for CVS.

Parameter	Bugs	SuperBugs	HyperBugs	Full Sample	(Parameter Description)
#Bugs	25965	650	290	16924	
Intercept	(−)***	(+)***	(+)***	(−)***	
tpsStart^{BC}	(−)**			(+)***	(first commit − opening date)
nauth^{C}	(+)***	(+)***	(+)***	(+)***	(maintainers mentioning bug)
nfile^{C}	(−)***			(−)***	(files touched by bug-commits)
ncome^{C}	(+)***			(+)***	(commit-comments per bug)
ncomi^{C}	(+)***			(+)***	(commits per bug)
sadd^{C}	(−)**	(−)**		(−)**	(added lines of code per bug)
sdel^{C}		(−)**		(+)**	(removed code per bug)
ccsz^{B}	(−)***			(−)***	(# addresses in cc-list)
attac^{B}		(−)**			(attachments per bug)
patc^{B}		(+)***			(attachments marked “patch”)
depen^{B}	(−)***	(−)**		(−)***	(bug dependencies)
bloc^{B}	(+)***	(+)**		(+)***	(bug blocks)
comm^{B}					(number of comments)
comau^{B}	(+)*			(+)***	(# commentators)
comli^{B}					(# lines of comments)
priority^{B} 0	(−)***	(−)*		(+)***	
priority^{B} 1	(−)***	(−)*		(+)***	
priority^{B} 2	(−)***	(−)**		(+)**	
priority^{B} 3	(−)***			(+)***	
priority^{B} 4	(−)***		•		
priority^{B} 5	•	•	n/a	•	
severity^{B} 1					
severity^{B} 2	(+)***		(+)**	(+)*	
severity^{B} 3	(+)**				
severity^{B} 4	(+)***		(−)*		
severity^{B} 5	(+)***			(−)**	
severity^{B} 6	•	•	•	•	

at least compared to all bugs that do not depend upon any other and specially bugs that are not part of a bug network.

Third, the number of developers (nauth) contributing to the code is positively related to survival time. Resistant bugs to which more numerous and different commits are related (ncome, ncomi) also tend to be fixed less rapidly, and similar effects hold for resistant bugs when more numerous people participated in the bugzilla discussion (comau). On the contrary, tpsStart, i.e. the length of the discussions and experimentations before a first patch is applied to the main branch, reduces the survival time of resistant bugs. Similarly, ccsz (number of developers who were copied when updates were made to the bugzilla system) also reduces the survival time of resistant bugs. Both of these effects disappear for superbugs. On the full sample, ccsz is also significant, and still negative, while tpsStart is significant, but positive. Evidence is therefore mixed here about the effect of "eyeballs" on fixing time. The number of active participants in the discussion tends to slow down fixing, but this might just reflect how complex to fix a bug is. Conversely, longer discussions (tpsStart) and the number of observers (ccsz) would tend to allow for a solution to be found more rapidly at least for resistant bugs.

Fourth, the fixing of superbugs is not affected by most of the variables that influence fixing resistant bugs. On the contrary, they are sensible to the number of lines deleted — maybe as a consequence of simply removing the part of the code that created a superbug? — and also to the number of attachments and to the number of patches sent in the Bugzilla discussion: the higher the number of patches sent, the longer it takes to fix a superbug; conversely, the higher the number of attachments, the lower the survival time of a superbug. A straightforward interpretation for the former finding is that the number of patches might be a consequence of how difficult it is to fix a given superbug. About the puzzling latter finding, it should be noted that attachments are often screen captures and other contextual elements: a superbug would then tend to be fixed more rapidly as soon more elements of context would be contributed to the discussion. This would be for instance in line with the fact that intermittent failures generally tend to be difficult engineering problems, whose intermittence is often due to missing *contextual* elements. An open question here is whether the provision of contextual elements earlier during the bugzilla discussion would have prevented the transformation of normal bugs into superbugs.

Conclusion We believe that understanding how superbugs could be fixed more rapidly could be of special relevance vis-à-vis the reliability of open-source software. priority and severity variables as currently defined in bug repositories do not appear optimized yet in this respect. We suggest that analyzing the formation of bug report networks, clarifying the nature of the discussion in bug repositories between participants assuming different roles, and understanding how contextual elements are brought into bug repository discussions are also interesting research avenues.

References

1. K. Crowston, J. Howison, and H. Annabi. Information systems success in free and open source software development. *Software Process*, In Press.
2. K. Crowston and B. Scozzi. Coordination practices for bug fixing within FLOSS development teams. In *Proc. CSAC*, 2004.
3. A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development. *ACM Trans. Softw. Eng. Methodol.*, 11:309–346, 2002.
4. E. S. Raymond. The cathedral and the bazaar. *First Monday*, 3, 1998.
5. R. J. Sandusky, L. Gasser, and G. Ripoche. Bug report networks. In *Proc. ICSE Workshop Mining Software Repositories*, 2004.

Free/Open Services: Conceptualization, Classification, and Commercialization

G.R.Gangadharan[1], Vincenzo D'Andrea[1] and Michael Weiss[2]

[1] Department of Information and Communication Technology, University of Trento, Via Sommarive, 14, Trento, 38050 Italy
{gr,dandrea}@dit.unitn.it
[2] School of Computer Science, Carleton university, 1125 Colonel By Drive, Ottawa, K1S 5B6, Canada
weiss@scs.carleton.ca

Abstract. The concept of Free/Open Services (F/O-Services) emerges by bringing together services with Free/Open Source Software (FOSS). F/O-Services enable the creation of transparent composite services collectively and allow people and other services to access them. This paper extends the concept of F/O-Services beyond the level of open interfaces, analyzing the associated licensing interpretations and exploring the notion of open service dependencies. Further, the paper overviews the business models for F/O-Services as a part of this social mechanism of exchange.

1 Introduction

Software has, traditionally, been perceived as a product, requiring possession and ownership, in order to receive the desired performance. The common model for software use is to install and execute on a computer owned by the user or his/her organization. This model is overridden by Software-as-a-Service [1], a mechanism for renting software where users subscribe to the software they use. Service oriented computing (SOC) allows the software-as-a-service concept to expand to include the delivery of complex business processes and transactions as a service, allowing applications to be constructed on the fly and services to be reused everywhere and by anybody [2]. The two important motivations for opening interfaces through services are as follows:

- The trend toward componentization and commoditization of business functionality [3, 4] means that a component-based business will focus on its value-added functionality, and outsource non-value-added functionality.
- Opening interfaces leverages external innovation, as amply demonstrated by the great variety of mash-ups built using the Google Maps interfaces [5].

In general service consumers have no access to the implementation details of a service, including whether or not a service uses other services, and what are these other services. An approach similar to Free/Open Source Software (FOSS) that opens the availability and accessibility of the source code of services would significantly enhance the understandability of the service composition process (including data and control flow) and allow the creation of derivative services.

Free/Open Source Software (FOSS) is an encompassing term for the development models, legal terms, and sociological issues associated with this novel software paradigm [6].The acronym FOSS combines the views of Free Software and Open Source Software and implies the commonalities between these approaches. Inspired by the success of FOSS, we have conceptualized and analyzed the implications of Free/Open Services (F/O-Services) in [7]. In this paper, we extend our previous work by adding service dependencies to the notion of F/O-Services and elaborate the explicit expression of dependencies in licenses.

2 Free/Open Services

A service is represented by an interface part defining the externally visible functionality (and typically some non-functional properties [8]) and a realization part implementing the interface [9]. Generally a service would be available when invoked by another service/entity, but remains idle until the request arrives. Services provide universal interoperability, manifested by the web-like network of services created by the composition of lower level services into higher level services [10]. Composite services could be created dynamically based on functional and non-functional requirements. Individual services can be replaced in case of malfunctions or due to the changes of requirements. A truly dynamic service oriented system can achieve software evolvability. The dynamics and composability of services are at the core of service orientation[1].

The opaque nature of services often hides the details of operations from the service consumer. The consumer could neither see anything beyond the interface nor understand about the services being composed in a composite service.

A F/O-Service is inspired from FOSS concepts and is characterized by the following principles [11]:

- The source code of the interface (WSDL descriptions) as well as of the implementation should be available.
- The service should be allowed for modification and the modified services should be freely distributable.
- The service should allow derivation and should be freely reimplementable and executable.

A F/O-Service allows the access to the source code of interface as well as its realization, making composite services and derivative services. We extend the F/O-Service philosophy by introducing the term dependency.

[1] In this paper, we do not explore the implications of dynamic nature of service composition.

We define dependency between services as the description of the interactions of a service with other services. Interactions do not have a direction per se, but a dependency does. A dependency link is directed from the service user to the service provider. Consider a service A, which composes the services B, C, and D. Further, these services compose E, F, G, and H. Given the service A, we could not understand what the services are being composed in A. If we make

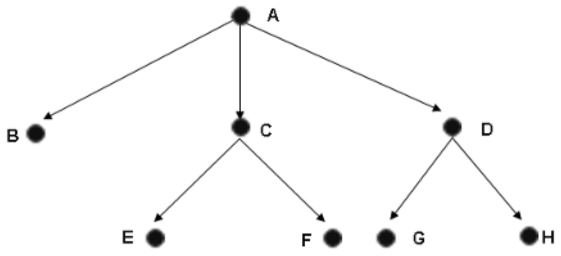


Fig. 1. Dependencies of a service

the dependencies of services open, we could achieve a service, whose service internals are completely exposed to the consumer. In Figure 1 circles and arrows represent services and their dependencies. From the given dependency graph, we could recognize the complete hierarchy of composed services. This approach is quite similar to white box description of components [12].

There are at least two notions of openness in the context of dependencies[2]:

1. The service declares which services it uses, but this does not imply a right for the consumers to invoke those services directly, if the service provider wishes to protect the intellectual property inherent in the composition and
2. The service allows others to reuse the relationships with other services it has. Restrictions imposed by the component services apply, of course.

These notions are not all-encompassing. For instance, there may be intellectual property rights attached to the selection of services during composition.

3 Free/Open Services Classification

Like software, a service is also an asset transferring an inherent value from the provider to the recipient. A service is a self-contained implementation of specific operations with a well-defined interface. However, the nature of services [7] precludes the direct adoption of software licenses for services.

Copylefting is the process of imposing copyright law to remove limitations on distribution and modifications, requiring to preserve the same freedoms in

[2] Opening dependencies implies only the provision of a list of composed services, and differs from fully exposing the application logic used to compose them.

the modified versions. From the perspective of service providers and developers, copylefting of services could be seen as a restriction imposed on the new service, that allows the value addition solely with the same conditions as the original. However, from the perspective of the service consumer, copylefting could be viewed as an ultimate guide for using any value added services inheriting from a particular copylefted service.

Free/open source software, in general, could be either copylefted or not. However, in addition to the dimension of source code, F/O-Services have another dimension: execution/usage. Unlike software, services are not resident in the recipient's environment. Though FOSS licenses do not discriminate in the uses of a software, the dynamic binding and execution of services could enforce certain restrictions for the execution/usage of F/O-Services. These restrictions could be of several kinds, for example,

- a service should be allowed to be executed for a certain number of times;
- a service should be used for certain purposes (for example, academic use);
- a service should require payment from the consumer for execution.

Now, considering the continuum of execution/usage with respect to copylefting in association with the openness of source code, we classify the F/O-Service licenses as follows[3]:

Unbounded Licenses belong to the most permissive family of F/O-Service licenses. These licenses allow licensees to use the service (and its source code) for any purposes without any restrictions. These licenses are wholly unrestricted for any kind of value addition of services.

Disjoint Licenses require no copylefting on any value addition but imposing restrictions on usage/execution of the service. These licenses are disjoint (unrelated) from the licenses of parent services.

Confined Licenses are the family of licenses where execution/usage may be restricted and could allow any kind of value addition, provided the value added services could be licensed under the same family. These licenses enrich the F/O-Services community.

Accessible Licenses refer to the family of licenses where execution/usage may be unrestricted and could allow any kind of value addition, with the requirement of copylefting.

The taxonomy of F/O-Service licensesis tabulated as shown (see Table 1):

Table 1. Taxonomy of F/O-Service Licenses

	Execution Restricted	Execution Unrestricted
Copylefting	*Confined Licenses*	*Accessible Licenses*
Non copylefting	*Disjoint Licenses*	*Unbounded Licenses*

[3] We support the rights of a service provider to make use of any license for their service, and highly recommend that service providers obtain appropriate legal advice regarding their selection of a service license.

4 Free/Open Service Business Strategies

Business strategies are the specifications of complex real world descriptions for managing a business by an organization in a sustainable way. There exists a wide range of business models for FOSS [13] as well as for services [14].

A service may be available at no cost. However, it could motivate the consumer to purchase something. Like traditional commercial software, services could begin their product life cycle as closed and then could become open when appropriate. Further, F/O-Services could adopt a complementary service/product scheme where revenue comes from media distribution, branding, training, consulting, and custom development. Also, by following the dual licensing strategy, a service can be licensed under both an open source inspired license and a proprietary license.

Service hosting is another business strategy for F/O-Services. A F/O-Service provider could host the services defined by others, thus making a viable business opportunity. A service host provides the capacity for executing a F/O-Service.

The intermediary and shared infrastructure models [15] can also be adapted to F/O-Services. One type of intermediary is a service aggregator. It adds value by composing other services so that a new functionality arises that was not available before. Intermediaries may also add value through the pre-selection of component services and managing their quality. A shared infrastructure service is an open service jointly developed by service users or providers for their common usage. In this case, it is more economic to share the development costs rather than developing the capabilities provided by the services individually.

5 Concluding Remarks

Though the standards of services are open, the domain of SOC is not enjoying the freedom of openness till now. As freedom of distribution and freedom of modification are the core principles of free and open source licensing, we think an approach inspired by FOSS for conceptualizing the service licenses would be beneficial to the services community. To the best of our knowledge, [11] is the first published work heralding the rise of F/O-Services. Following this work, very recently, there are few informal blogs or writings scattered in the Internet [16, 17]. However, these works neither conceptualize F/O-Services completely nor intend to formalize the representation of licenses for F/O-Services. exploring the concept of dependencies.

Exposing the service dependencies could matter to the service consumer for a number of reasons, such as the consumers could deny a service that composes a service from a particular service provider. Service providers fail to get sufficient guidance for the implementation from the inadequate interfaces (the partial information conveyed through current service interfaces) [10]. Similarly, service consumers could make incorrect assumptions about the service implementation given these inadequate interfaces. The opening of service increases the quality of service integration by reducing the number of composition errors due to misinterpretation of service interfaces, and failures due to hidden side-effects, thus reducing the cost of developing and maintaining composite services.

References

1. Elfatatry, A., Layzell, P.: Negotiating in Service Oriented Environments. Communications of the ACM **47**(8) (2004) 103–108
2. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services Concepts, Architectures, and Applications. Springer Verlag (2004)
3. Sanford, L., Taylor, D.: Let Go to Grow: Escaping the Commodity Trap. Prentice Hall (2005)
4. Cherbakov, L., Galambos, G., Harishankar, R., Kalyana, S., Rackham, G.: Impact of Service Orientation at the Business Level. IBM Systems Journal **44**(4) (2005) 653–666
5. Mulholland, A., Thomas, C., Kurchina, P.: Mashup Corporations: The End of Business as Usual. Evolved Technologist Press (2006)
6. Feller, J., Fitzgerald, B.: A Framework Analysis of the Open Source Software Development Paradigm. In: Proc. of the 21st Annual International Conference on Information Systems. (2000) 58–69
7. D'Andrea, V., Gangadharan, G.R.: Licensing Services: The Rising. In: Proceedings of the IEEE Web Services Based Systems and Applications (ICIW'06), Guadeloupe, French Caribbean. (2006) 142–147
8. Wang, G., MacLean, A.: Software Components in Contexts and Service Negotiations. In: CBSE Workshop. (1999)
9. Papazoglou, M., Georgakopoulos, D.: Service Oriented Computing. Communications of the ACM **46**(10) (2003) 25–28
10. Weiss, M., Esfandiari, B., Luo, Y.: Towards a Classification of Web Service Feature Interactions. Computer Networks **51** (2007) 359–381
11. D'Andrea, V., Gangadharan, G.R.: Licensing Services: An "Open" Perspective. In: Open Source Systems (IFIP Working Group 2.13 Foundation Conference on Open Source Software), Vol. 203, Springer Verlag. (2006) 143–154
12. Szyperski, C.: Component Software: Beyond Object Oriented Programming. ACM Press, New York (1998)
13. Raymond, E.: The Magic Cauldron. http://www.catb.org/ esr/writings/magic-cauldron/magic-cauldron.html (1999)
14. Hagel, J.: Out of the Box: Strategies for Achieving Profits Today and Growth Tomorrow Through Web Services. Harvard Business School Press (2002)
15. Weill, P., Vitale, M.: Place to Space: Migrating to E-business Models. Harvard Business School Press (2001)
16. Slashdot: Web Services and Open Source at OSCON. http://developers.slashdot.org/article.pl?sid=06/07/26/1537213 (Posted on July 26, 2006)
17. log.ometer.com: Log for July, 2006. http://log.ometer.com/2006-07.html (Posted on July 29, 2006)

Surveying Industrial Roles in Open Source Software Development

Øyvind Hauge, Carl-Fredrik Sørensen, Andreas Røsdal
Norwegian University of Science and Technology (NTNU), 7491
Trondheim, Norway

Abstract. Industry uses Open Source Software (OSS) to a greater and greater extent. We have defined four industrial OSS roles; OSS provider, OSS integrator, OSS participant and Inner Source Software (ISS) participant. Based on these four roles we have performed a survey in the ITEA COSI project. We provide initial answers to what motivates companies to undertake these roles, what are the advantages and challenges of undertaking them, and which development practices they use while undertaking these roles.

Key words: Open Source, Industry, Roles, Survey, Motivations, Development Practices

1 Introduction

The cost of producing software from scratch goes hand in hand with the steadily increasing size and complexity of the software. Reuse of standard components has been seen as one solution to keep costs down. Reusable components have been developed in-house or acquired from other vendors.

OSS provides quality software, enables new ways of developing software, and makes new business strategies possible. OSS can be important in the battle against constantly larger and more complex software. Several major industrial actors like Sun Microsystems, Oracle, IBM, and Novell, have already started to benefit from OSS.

The entry of industry into the OSS field opens up a new research arena. The ITEA COSI project wants to increase the understanding of how industry can benefit from OSS. As part of the ongoing work in the ITEA COSI project we have performed a survey of current OSS development practices in parts of the European software industry. The survey gave several interesting indications. The availability of OSS is perhaps the most important reason behind use of OSS. The main advantages for a company having an OSS product come from, value added by supplementary products and community innovation. Attracting and supporting an OSS community requires hard work and there are challenges related to community contributions.

We start by presenting the four industrial OSS roles and the applied research method before we present our results and sum up with a discussion and conclusions.

Please use the following format when citing this chapter:

Hauge, Ø., Sørensen, C.-F., Røsdal, A., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 259–264.

2. Related work and Industrial Roles

Our literature survey did not discover many empirical studies of industrial OSS involvement. However, examples can be found e.g. [1-5].

We want to highlight the need for more varied and reproducible empirical research. The majority of the publications we found were case studies or experience reports which are hard to reproduce. The work is in many cases performed in only one setting, most often in a non-industrial setting.

Based on literature and conversations with the industrial partners of the ITEA COSI project we defined four industrial roles: OSS Provider, OSS Integrator, OSS Participant, and Inner Source Software (ISS) Participant.

An *OSS provider* is a company which controls the code base of an OSS product. MySQL, Trolltech, and Sun Microsystems are some examples. The *OSS integrator* is a company which, uses OSS components in their products or build their products on top of OSS infrastructure. The *OSS participant* is a company actively interacting with one or more OSS projects. IBM and SUN are for instance participating in the development of the Apache DB. The *ISS participant* is a company participating in an inter department or inter company collaborative development using OSS development practices.

3. Research Method

In the first phase of the ITEA COSI project, we wish to create a baseline description of the industrial OSS related development. The following questions were based on a literature review and in conversations with project partners: Why do industrial actors undertake the four OSS roles? What are the advantages and challenges related to undertaking them? Which development practices are used in these roles?

Based on these questions, we created an interview guide which was used in semi-structured interviews with Norwegian COSI partners. The interviews were performed at the offices of the industrial partners and all of them were recorded and later transcribed.

We interviewed two developers in company A, one developer in company B, and one developer and one CEO in company C. Company A is a small company which uses OSS in their development. Company B is a medium sized consulting company delivering services and products based on OSS. Company C is a medium sized company which provides an OSS product.

The interview guide and the results from the interviews were used as a basis for a web-survey. The survey had one part for each OSS role.

The ITEA COSI project consists of big companies from telecom and embedded software, but also smaller and more traditional software companies. Selection of the respondents was because of the composition of the project, unfortunately out of our hands. We distributed the survey to the all of the project partners and encouraged them to respond at least once. The companies selected their respondent(s) themselves and we received the following number of responses; OSS provider: 3, ISS participant: 6, OSS participant: 6 and OSS integrator: 9, in total 24 responses.

4. Results

OSS providers are motivated to release their products as OSS of several factors. The community can perform testing and provide new functionality, bug-fixes, bug-reports, and translations. This may enhance the functionality and increase the quality of the product. The community members may contribute to the innovation of the product in form of new ideas and new requirements. They can also provide supplementary products and services.

Releasing a product as OSS is a way to make it available to a large user group. If the community is satisfied with the product, it will most likely share its experiences with others and thereby give the OSS provider free marketing and increased publicity.

Increased value, availability and publicity, boost the possibility of attracting new users. This is important because many industrial OSS providers sell services related to their OSS products. The more users, the more potentially paying customers and the more likely it is that someone will contribute to the development of the product.

We believe that the innovation and the supplementary products and services which increase the value of the product are more important than code contributions. This is because the Oss provider has to review contributions in form of code, requests, and opinions.

Maximizing community contributions and reducing the work related to these contributions is one of the challenges an OSS provider faces. Attracting a community is another major challenge for an OSS provider and according to our respondents, hard work.

It is important to offer the community a piece of quality software they need, infrastructure to support the community, enough documentation and information to get the community members going and to make them feel involved. However, it is important not to involve the community too much because involvement will create overhead and delays.

The OSS integrator is motivated by the low purchase price of the OSS products. Perhaps even more important is the high availability of OSS. Standard compliance was also mentioned as a reason why people use OSS.

Many OSS products are available through project web sites containing documentation, forums and mailing lists, bug and feature trackers, road maps, developer info and so on. The honesty about the true status of the OSS product and the availability of information make it easier for the OSS integrator to understand and evaluate it.

OSS components are primarily selected through informal processes. The OSS integrator discovers a need for a component. He forms an initial idea of what the software should do. Based on these initial requirements he performs an informal search to create a long-list. This long-list is later reduced to a short-list. The components on the short-list are tested or evaluated closer before one product is selected.

The candidate components may be found through many sources; prior experience, friends or co-workers, request for help on forum or mailing-list, searches in OSS portals or search engines. Search engines are used to find both single components and comparisons of several components.

Missing functionality, incompatible licenses, unfamiliar programming languages, lack of stable releases, no activity in community, bad or no reputation, and absence of documentation, are easy-to-check evaluation criteria. To evaluate the components further the developer may subscribe to mailing lists, study documentation, perform code reviews, and test the software in a small prototype. Plans and roadmaps, compatibility to other software, standard compliance, reputation of the product and the provider, the development process used in the community, and support from community or a commercial provider, were all mentioned as evaluation criteria in this process. This evaluation was mostly informal but some respondents reported that they used checklists.

The OSS integrator is faced with some challenges. There are vast numbers of OSS available out there and finding quality products can be hard.

By changing the source code of the OSS products he uses, the OSS integrator is left with two choices: He can keep the changes to himself or feed the changes back into the product. Convincing the OSS project to include these changes can be hard. If he is unable to make the OSS project include his changes he has to maintain this code himself. This could be time-consuming and it may lead to problems with new releases of the OSS.

Most of the **OSS participants** could not surprisingly be classified as active or passive users. They provide occasional bug fixes and requirements, subscribe to mailing-lists, read news, and primarily use the software.

The respondents were overall satisfied with the OSS products, their communities, information from the community, and their relationship with the community. However, they acknowledged that they would have been able to influence the community more through increased participation.

Participation as a company was not surprisingly rooted in the need for the product. Learning was also mentioned as one important motivation for some companies. On the individual level learning, idealism, and personal interest in the product were mentioned as the most important factors.

The participants in ISS development use some development practices often used in OSS development. The use of e-mail and mailing list was due to the distributed development quite extensive.

To provide the participating developers a shared view of the code, code repositories were used. These repositories were controlled by gatekeepers or module owners. Based on the code base, several pre-releases of the software were made available to give the users an early impression of the product and to allow the users to provide feedback to the developers.

Some of the respondents reported saved development effort and maintenance effort due to ISS cooperation.

5. Discussion and conclusions

In the section about related work we requested more and more varied empirical research related to industrial OSS involvement. We are aware of some of the limitations of our own work and we will discuss some of these here.

The survey was intended to be a baseline for the companies in the ITEA COSI project. The selection of respondents was done from this population and we cannot claim that our results are valid for other populations.

The number of respondents was unfortunately quite low. The selection of respondents was done by convenience sampling. We were, due to the sampling method, unable to control mortality rates and drop out rates for the questionnaire. These factors reduce the internal validity and the statistical validity of the survey.

We have however increased the validity through interviews with some of the respondents and through expert review. We have presented the results to the ITEA COSI project and to several of the respondents. None of them gave us any indications that the results were flawed.

We believe that our work is a step on the way to understand how industry can benefit from OSS products and development methodologies. The survey has given us initial ideas of what motivates companies to undertake the four roles OSS provider, OSS integrator, OSS participant, and ISS participant. Furthermore, we have described some of the advantages and challenges related to undertaking these roles. At last we have started to describe some of the processes and practices used by these roles.

The work of answering the initial questions about motivations, processes, advantages and challenges are by far not completed. We will continue this work and a second version of the survey is under development. This survey will be distributed to a larger European population through ITEA.

Acknowledgement

The Norwegian COSI is sponsored by the Norwegian Research Council's IKT-2010 program. The COSI project is part of the ITEA 2 program.

References

1. W-G. Bleek, M. Finck, and B Pape, Towards an Open Source Development Process? Evaluating the Migration to an Open Source Project by Means of the Capability Maturity Model, *Proceedings of the First International Conference on Open Source Systems*, Genova, Italy, 37–43 (2005)
2. C. Jensen and W. Scacchi, Collaboration, Leadership, Control, and Conflict Negotiation and the Netbeans.org Open Source Software Development Community, *Proceedings of the 38th Annual Hawaii International Conference on System Sciences,* 196b-196b, (2005).
3. V. K. Gurbani, A. Garvert, and J.D. Herbsleb, A Case Study of a Corporate Open Source Development Model, *Proceeding of the 28th international Conference on Software Engineering ICSE '06*, Shanghai, China, 472–481 (2006)
4. C. Rossi and A. Bonaccorsi, Why Profit-Oriented Companies Enter the OS Field? Intrinsic vs. Extrinsic Incentives. *Proceedings of the fifth Workshop on Open Source Software Engineering*, 1–5 (2005)
5. L. Dahlander and M. G. Magnusson, Relationships between Open Source Software Companies and Communities: Observations from Nordic Firms. *Research Policy*, 34(4), 481–493 (2005)

Guiding the Discovery of Open Source Software Processes with a Reference Model

Chris Jensen and Walt Scacchi
Institute for Software Research, Bren School of Information and Computer Sciences, University of California. Irvine, CA 92697-3440 USA
{cjensen, wscacchi}@ics.uci.edu
WWW home page: http://www.ics.uci.edu/~{cjensen, wscacchi}

Abstract. This paper describes a reference model for open source software (OSS) processes and its application towards discovering such processes from OSS project artifacts. This reference model is the means to map evidence of an enacted process to a classification of agents, resources, tools, and activities that characterize the process.

Keywords. Reference model, open source, process discovery

1 Introduction

OSS community observers and novice members often face the challenge of determining how the community works and, equally important, how to properly contribute. Process discovery aims to answer these questions. As researchers and individuals wishing to understand and/or participate in OSS communities, process discovery can be a time consuming task as each community has its own way of doing things. Much of this time is spent collecting and coding evidence, as in any qualitative data analysis. As with errors committed early in the software development lifecycle, data miscoded early in the overall research process has a high cost. We use a reference model based approach for process discovery to assist in coding process evidence to reduce the risk and the cost associated with such coding errors, as well as to more completely articulate discovered OSS processes [1, 2, 3]. The discovery and codification of OSS projects is also a prerequisite to continuous improvement of these processes.

Our reference model serves as a guide to coding process data collected from project artifacts. However, as our reference model is based on a process meta-model, [4], the reference model can serve as a guide to help in coding and modeling OSS processes in forms suitable for automated analyses [9]. Our hypothesis is that a reference model-based approach provides a systematic approach to the problem, giving us consistent and satisfactory results when applied to multiple projects and different types of processes.

In the remainder of this paper, we define a reference model for OSS processes and demonstrate its use in guiding process discovery. We give examples of its usage and conclude with further research opportunities.

2 Reference Model Composition

Our reference model provides a mapping between process evidence discovered by searching the project Web and a classification scheme of process attributes. To do this, we must to determine what aspects of the process we wish to discover. These can be defined by our process meta-model. This meta-model is neither specific to our domain (OSS development processes), nor software processes. The meta-model we use is that of Mi and Scacchi [4]. It establishes a vocabulary of process modeling objects, attributes, relations, and constraints that describe processes. This meta-model defines processes by hierarchical decomposition and by precedence (partially ordered resource flows): processes are composed of tasks (sets of related actions) and atomic actions. Each action is defined in terms of the following process entities: agents in different roles participate in the process activity using tools that require (consume) or provide (produce) resources (artifacts). Tool-based actions are coded using tool invocation scripts/methods that may contain narrative description[s] of the action, or "HTML markup specifying a form to be completed as part of the task. We find these to be a minimal set of concepts necessary to describe a process [9, 10]. This meta-model is augmented with control-flow grammar in the process modeling language we use to formally represent software processes, showing the order in which activities are instantiated [cf. 9].

In practice, we have a dictionary of terms observed in these projects, correlated with known associations from the taxonomy of actions, agents, tools, and resources we constructed based on the original case studies. Tool invocation scripts have been left to downstream process discovery tasks due to their complexity.

Whereas existing ontologies for OSS development (e.g. what Simmons and Dillon [5] presented at OSS 2006 and Rothfuss's [6] framework for open source projects) provide a classification of OSSD concepts, they lack mappings to known process entity instances required to discover software processes. It is worth noting that an individual term in our dictionary may correspond to several different types of process entities in the taxonomy, both within a main branch (e.g. tool) and across main branches (e.g. tool or resource) depending on the context of the term's usage. As an example, consider the term "report" in Figure 1.

```
Term: report
  Known Actions: testing, defect reporting, logging
  Known Agents: none
  Known Resources: whitepapers, test results, web
     server logs, defects, feature requests
  Known Tools: defect repository, test suite
```
Figure 1. Example reference model mapping

There are many OSS artifacts that encode process information and their presence varies by community. Some of the more common artifact include webpages [10], chat transcripts [7], defect reports, source repositories, development and community infrastructure tools [8], and development resources, including process fragment descriptions (e.g., How-To guides, FAQs, etc.) [7]. These artifacts encode data useful for process discovery in four dimensions: s*tructure* (how project-related software development artifacts are organized), *content* (types of artifacts and information they contain), *usage patterns* (user interaction within the community web), and *update patterns* (content updates, including initial creation and deletion).

OSS artifacts vary along these four dimensions over time, and this variance is the cause or consequence of process events. By observing patterns of patterns of reference model attributes within a process iteration and how these patterns change across iterations, we can discover different types of processes and their evolution over the life of a project. We give some examples of this in the next section where we discuss our experiences in applying our reference model in the course of process discovery.

3 Experiences in Reference Model Based OSS Process Discovery

We have applied this technique to discovery of three different types of processes in and around the NetBeans IDE, Apache HTTPD server, and Mozilla projects for a total of seven case studies. We have framed our research with the perspective of an observer or would-be contributor [9] and have, thus, limited ourselves to data that is publicly available via their respective online project portals. Consequently, we have found that access to all four artifact dimensions that encode process data has been limited, especially usage patterns. These limitations have varied across projects, but also between artifacts within a project. Although our resulting process models may have been more precise if not for these limitations, each study, nevertheless, yielded an overwhelming amount of data to examine. In this section, we describe how we have used our reference model in each of the three types of processes we looked at.

3.1 Development Processes

Surveys of development processes in Apache, Mozilla, and NetBeans seeded our initial reference model. These studies examined the requirements and release processes in NetBeans, the Apache server's release process, and the Mozilla testing cycle, circa 2002. The processes were discovered without an explicit reference model, but rather drew on our knowledge of the process meta-model [4] and the projects under study. From these studies we constructed a taxonomy of types of agents, tools, resources, and actions observed and built our reference model dictionary using instances of these entities from project data, as discussed above. Successive process discovery case studies have further enhanced the reference model's precision and relevance in guiding subsequent OSS process discovery,

3.2 Interorganizational Process Communication

In our second case study, we examined interorganizational process communication [10] across several projects (including, but not limited to NetBeans, Apache, and Mozilla) that form a software ecosystem. We said processes that communicate activities or synchronize resources across OSS projects are "*integrative* if they identify compatibilities or potential compatibilities between development projects" (i.e. enables external stakeholders to continue following their internal process as normal) or *conflictive* if "the degree of accommodation or adaptation becomes too great" [10].

In the course of discovering communicating processes, we examined relationships that synchronize resources shared between projects. Some of the most common relationships were integration of tools and libraries produced by one project and used by another and the participation of individuals on several projects, sometimes called linchpin developers [11]. The reference model produced in the course of discovering development processes provided insufficient detail in terms of proper names of specific tools and resources (especially shared libraries) necessary to observe the types of interorganizational relationships that precipitated integration and conflict. Moreover, discovering interorganizational processes requires tracking project specific vocabularies in order to identify instances of process communication between projects. Specifically, we had to denote producers and consumers of specific libraries and development tools, as well as project contributors. As a complicating factor, process communication often occurs outside project web portals, in other channels, both public and private. Some of the richest data available regarding interorganizational process communication between the NetBeans project and the Eclipse IDE project (see http://www.eclipse.org), for example, came from interviews and reports regarding private meetings between the governing body of the Eclipse project and the management of SUN Microsystems, employer of many developers and project leaders for the NetBeans project. Nevertheless, evidence of process communication, and more so the extent to which it was integrative or conflictive, was difficult to observe.

3.3 Role Migration Processes

Our third set of case studies [7] involved observing OSS project participants changing roles. We observed several different tracks of participation, ranging from source code development to community governance and how developers change roles over the course of their participation (or career) within an OSS project.

The key to observing role migration lies in changes in the activity patterns for individual project participants. Using our original reference model as a basis, we saw that the types of actions an individual is associated with have shifted to other branches of the action taxonomy over time. A shift to a closely related branch indicated a migration along the same participation track while a shift to a distant branch indicated a shift to a different track. This technique also allowed us to determine individuals with multiple roles, as well as to detect adoption and abandonment of such roles over time. Using this logic, if many participants show the same migration patterns near the same point in time, we may infer a shift in the participation track-specific process. In this event, we would not be surprised to find shifts in the resources and tools along that track, though may not always be the case.

4 Discussion and Future Work

Becker-Kornsteadt [12] gives a detailed model of process discovery, used in conjunction with the Spearmint project. Abstracting the problem to three stages: data collection, data analysis, and data presentation, the reference model offers only a partial solution to the data analysis stage. After data is coded, it must be assembled into a set of actions (composed of sets of participating agents, tools, and resources required and created by the action, and invocation scripts, as appropriate). Lastly, to produce a process description, this set of actions must be arranged in a fashion representing the (partial) ordering in which they took place [13].

We have looked towards automation to reduce the substantial effort required for process discovery. Automatically coding process evidence is a tantalizing objective, given the advances in machine learning. The example in Figure 1 demonstrates the importance of context in precisely mapping process evidence to classification, suggesting that full automation is either not yet attainable or has a high cost. With this consideration, we have pursued an interactive direction, based on search technology. Our strategy is to query project artifacts using terms from our reference model and, where a term may have multiple mappings, allow the human process discoverer to make a determination as for which is accurate. This approach significantly reduces the effort in locating process evidence within project data and partially reduces the effort associated with data coding. Further, the approach is well suited for artifact-specific analysis and tracking timestamps associated with each document. These temporal signatures are necessary for establishing action-task-subprocess-process composition, as well as partial ordering of events. These efforts are in progress. Other research efforts [14, 15] have sought automated OSS process discovery, though these appear to focus on structure, usage patterns, and update patterns and less on their often semi and unstructured content.

The reference model cannot eliminate process discovery risk altogether, due to the variance in lexica between projects, however its usefulness as heuristic can be improved through updates and process (and project) specific tailoring. In this paper, we discussed using reference modeling in discovering OSS processes. We discussed the composition of such a reference model. We looked at where to find process data to apply the model to. We saw how it was used and evolved over the course of several case studies, and further opportunities for reference modeling to lower the risks and costs of process discovery.

Acknowledgments

The research described in this paper has been supported by grant #0534771 from the U.S. National Science Foundation. No endorsement implied. Mark Ackerman at University of Michigan, Ann Arbor; Les Gasser at University of Illinois, Urbana-Champaign; John Noll at Santa Clara University; Margaret Elliott and others at the UCI Institute for Software Research are collaborators on the research described here.

References

1 S. Koch, S. Strecker, and U. Frank, Conceptual Modelling as a New Entry in the Bazaar: The Open Model Approach. *Proc. Second Intern. Conf. on Open Source Software*, 8-10 June, 2006 Como, Italy (Eds) E. Damiana, B. Fitzgerald, W. Scacchi, and M. Scotto, p. 9-20

2 J. vom Brocke and C. Buddendick, Reusable Conceptual Models - Requirements Based on the Design Science Research Paradigm, *1st Intl. Conf. on Design Science Research in Information Systems and Technology*, Poster Paper, Eds: A. Hevner, Claremont, CA, USA, 24-25 Feb, 2006.

3 C. Jensen and W. Scacchi, Applying a Reference Framework to Open Source Software Process Discovery, in *1st Workshop on Open Source in an Industrial Context*, Anaheim, CA October 2003.

4 P. Mi, and W. Scacchi, A Meta-Model for Formulating Knowledge-Based Models of Software Development, *Decision Support Systems*, **17**(4), 313-330 (1996).

5 G. Simmons and T. Dillon, Towards an Ontology for Open Source Software Development. *Proc. Second Intern. Conf. Open Source Software*, 8 June, 2006 Como, Italy (Ed) E. Damiana, B. Fitzgerald, W. Scacchi, and M. Scotto, p. 65-76

6 G. Rothfuss, A Framework for Open Source Projects. Master Thesis in Computer Science, Department of Information Technology, University of Zurich, 2002.

7 C. Jensen and W. Scacchi, Process Modeling Across the Web Information Infrastructure. Software Process: Improvement and Practice, Special Issue on ProSim 2004, 10(3), 255-272 (2004).

8 T. Halloran and W. Scherlis, High Quality and Open Source Software Practices, *2nd Wrkshp. on Open Source Software Engineering*, Orlando, FL, 25 May, 2002.

9 W. Scacchi, Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings- Software*, **149**(1) 25-39 (2002).

10 C. Jensen and W. Scacchi, Process Modeling Across the Web Information Infrastructure. Software Process: Improvement and Practice, Special Issue on ProSim 2004, 10(3), 255-272 (2004).

11 G. Madey, V. Freeh, and R. Tynan, Modeling the F/OSS Community: A Quantitative Investigation, in S. Koch (ed.), *Free/Open Source Software Development*, (Idea Group Publishing, Hershey, PA. 2005), pp. 203-221.

12 U. Becker-Kornstaedt, Towards Systematic Knowledge Elicitation for Descriptive Software Process Modeling, *Intl. Conf. on Product Focused Software Process Improvement,* Kaiserslautern, Germany, 10 Sep, 2001 (Ed) S. Bomarius, Lecture Notes in Computer Science, Springer, 2188, pp. 312-325 (2001)

13 P. Feiler and W. Humphrey, Software Process Development and Enactment: Concepts and Definitions. *2nd Intl. Conf. on the Software Process: Continuous Software Process Improvement*, 28-40 (1993)

14 R. Sandusky, Software Problem Management as Information Management in a F/OSS Development Community. *Proc. 1st Intl. Conf. on Open Source Systems*, 11-15 Jul, 2005, pp. 44-49

15 Y. Liu, E. Stroulia, and H. Erdogmus, Understanding the Open-Source Software Development Process: A Case Study with CVSChecker. *Proc. 1st Intl. Conf. on Open Source Systems*, 11-15 Jul, 2005, 154-161

Effect of Coupling on Defect Proneness in Evolutionary Open-Source Software Development

A. Güneș Koru[1], Dongsong Zhang[2], and Hongfang Liu[3]

[1] Department of Information Systems, UMBC gkoru@umbc.edu
[2] Department of Information Systems, UMBC zhangd@umbc.edu
[3] Department of Biostatistics, Bioinformatics, and Biomathematics, Georgetown Medical Center hl224@georgetown.edu

Abstract. Previous research on closed-source software found that highly coupled software modules were more defect prone, which makes it important to understand the effect of coupling on defect proneness in open-source software (OSS) projects. For this purpose, we used Cox proportional hazards modeling with recurrent events. We found that the effect of coupling was significant, and we quantified this effect on defect proneness.

Key words: Open-source software, object-oriented software, defect proneness, coupling, Cox proportional hazards model, recurrent events, Mozilla.

1 Introduction

Coupling is the degree to which a program element is related to or interacts with other program elements. The higher the average coupling of elements in software, the more complex and defect prone it is considered to be [7]. The previous research on closed-source software has shown that highly coupled software modules are more defect prone compared to less coupled ones [2, 5]. Therefore, it is important to build statistical models to understand the relationship between coupling and defect-proneness in OSS.

However, the evolutionary aspects of OSS development processes require specialized modeling techniques. The structural characteristics of OSS modules (e.g., coupling) can change in the post-release period. Making the situation even more complicated, new modules can be added or some modules might be removed from a system shortly after measurement time. The traditional approaches to quality modeling, which measure specific system snapshots and relate them to future defect counts, cannot accommodate these special characteristics of OSS.

The main research contribution of this study is to develop and evaluate a statistical model in order to understand the effect of coupling on defect proneness while taking the dynamic aspects of OSS development into account. For this purpose, we adopted Cox proportional hazards modeling with recurrent events.

In the rest of the paper, we first explain our modeling approach and the data used in the study. Then, we present our modeling results. Finally, we discuss the implications of this work and conclude the paper.

2 Cox Proportional Hazards Model for Recurrent Events

Cox proportional hazards model [3] (henceforth *Cox model*) has become the most common technique used for various time-to-event analysis purposes in many fields [4, 8]. Cox model is connected to the counting process and Martingale theory [1], which makes it suitable for recurrent events. In recurrent event modeling, an event of interest is observed for a subject multiple times during a follow-up period [4].

In our study, each defect fix made to a class was considered an *event*. Being more defect prone meant having a higher risk of having events. We had a single time-dependent covariate, Coupling Between Objects (CBO), denoted by $x(t)$ below. CBO for a class C is defined as the number of methods and instance variables of other classes used by C.

We specify the *hazard function*, which is the instantaneous risk of an event for class i at time t, as:

$$\lambda_i(t) = \lambda_0(t) e^{\beta x_i(t)}. \tag{1}$$

β is the coefficient for $x_i(t)$ and λ_0 is an *unspecified* non-negative function of time called the *baseline hazard function*. It is the instantaneous hazard of having an event without any covariate effect, when $\beta = 0$.

Cox model is semi-parametric because it does not explicitly describe a baseline hazard function. It is proportional because the hazard ratio for two subjects would only depend on the differences in their covariate values. If one writes the right side of the Equation 1 for two subjects, say classes j and k, and takes their ratio, the result should be $e^{\beta(x_j(t)-x_k(t))}$, which is the instantaneous relative risk. Note that β should remain constant over time. This is an important assumption of any Cox model, known as proportional hazards assumption. We checked this assumption for our model (see Section 4). The details of the Cox model, such as the estimation of β, can be found in [4, 8].

3 Data for Recurrent Event Modeling

Table 1 presents hypothetical data for demonstration purposes. The subjects are classes, and the events of interest here are defect fixes. Each new class introduced to the system during an *observation period* is followed up until the observation period ends or until the class is deleted. Modifications made during the follow-up time are entered as *observation*s, which correspond to the rows in Table 1.

name	start	end	event	CBO	state
A	0	10	0	5	0
A	10	30	1	8	0
A	30	50	0	9	1
B	0	20	1	3	0
B	20	80	0	2	1
B	80	120	1	5	1
B	120	150	0	5	1
.	.	.	.	.	
.	.	.	.	.	

Table 1. Data Layout Used in the Study

Each modification creates a new observation with a $(start, end]$ time interval, where *start* is a time infinitesimally greater than the modification time; *end* is either the time of the next modification, or the end of the observation period, or the time of deletion, whichever comes first. The open bracket on the left and the closed bracket on the right mean that at any *end* time t, the observation that has t in its *end* column should be used in the internal computations of the Cox model. For example, for $t = 50$, the third row should be used. Open and closed brackets enable us to model non-overlapping observations. They carry no meaning about the timings of other data items, which are explained below.

When a class is added to the system, a new observation is entered with $start = 0$. The event cell is set to 1 if an event (defect fix) takes place at the time represented by *end*, or 0 otherwise. A class deletion is handled easily by entering a final observation whose event is set to 1 if the class is deleted for corrective maintenance, or 0 otherwise. CBO is a *time-dependent covariate* and its column carries the coupling measurements of the class at *start*. Its value may change during successive observations but can remain constant like a *fixed covariate* too. The *state* column in Table 1 is used to create a conditional Cox model. For any class, *state* is initially set to 0, and it becomes 1 after the class experiences an event, and always remains at 1 thereafter.

We developed Perl scripts to extract data from the CVS (Concurrent Versions System) of the Mozilla project between May 29, 2002 (with the release of Mozilla 1.0) and Feb 22, 2006, which was the observation period of our study. We obtained a complete measurement history of every single C++ class introduced to Mozilla during this observation period. The start and end times were computed in *minutes* based on the time tags of the CVS commits. The event data were obtained by automatically parsing the log portions of CVS commits and searching for the words 'defect', 'fix', and 'bug' in a non–case-sensitive manner to detect corrective changes. Our manual examination of 100 randomly collected CVS logs showed that the accuracy of the automated approach was 98%. Once a CVS commit was classified as a corrective change, the effected classes were determined with their most recent observations. The *event* field of those observations was set to 1. At the end, we obtained 15,545 observations that belonged to 4,089 classes.

4 Modeling and Results

The resulting conditional Cox model is shown in Figure 1. The model shows that CBO is highly significant with a very large z-statistic and a zero p value when entered using log transformation. This functional form of CBO was determined by inspecting the plots obtained by using the Poisson approximation [8]. The entire model is also very significant as shown by the Likelihood ratio, Wald, score, and robust score tests. Both normal and robust estimates show this significance. The coefficient for the $log1p(CBO)$[1] is 0.661, and its standard error estimate is 0.0117. The robust sandwich estimate of the standard error, which takes the intra-subject correlation into account, is 0.0297. Both of these standard error estimates are small, therefore, we can safely use $\hat{\beta} = 0.661$.

```
  n= 15545
            coef exp(coef) se(coef) robust se    z p
log1p(CBO) 0.661      1.94   0.0117    0.0297 22.3 0

           exp(coef) exp(-coef) lower .95 upper .95
log1p(CBO)      1.94      0.516      1.83      2.05

Likelihood ratio test= 3200  on 1 df,   p=0
Wald test            = 497  on 1 df,    p=0
Score (logrank) test = 3271  on 1 df,   p=0,   Robust = 148  p=0
```

Fig. 1. Modeling results using log CBO

There was no interaction between log CBO and time ($p = 0.93$). Therefore, the proportional hazards assumption of the Cox model was satisfied. An Arjas plot between the cumulative expected and cumulative actual number of events was drawn to see the overall fitness of the model. This plot closely followed the 45^o line, which showed good fitness. We also looked at the correlations between the expected and actual events. The Spearman's correlation was 0.77 and the Somer's D_{xy} rank correlation was 0.72. As a result, the model shown in Figure 1 has passed all the tests for a good fitting model.

5 Implications

The model in Figure 1 indicates that one unit of increase in the natural log of coupling caused Mozilla classes to experience a defect fix at a rate 94% higher.

[1] To accommodate $CBO = 0$, the natural log was taken after adding 1.

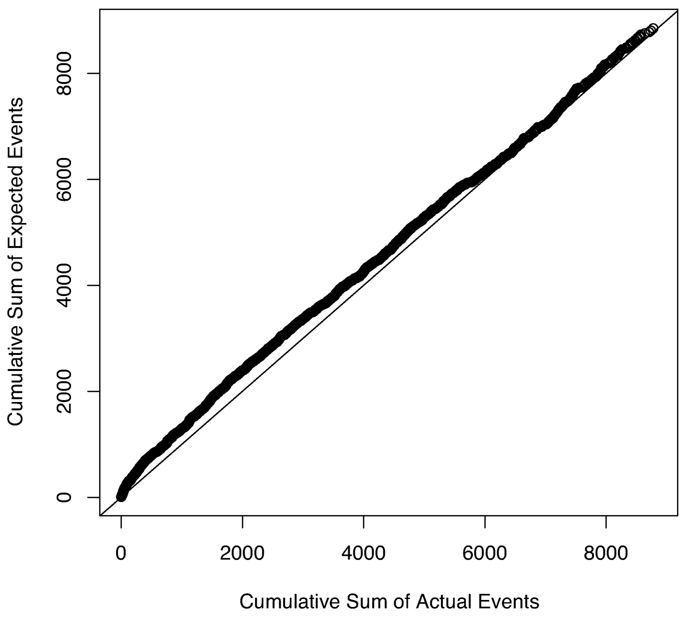


Fig. 2. Plot of cumulative sum of actual events versus cumulative sum of expected events

Our results have important implications considered the recent findings about the coupling in some OSS products.

Schach and Offutt [6] found that the degree of common coupling in the Linux kernel posed risks to the first release, and this situation deteriorated during the successive releases of this kernel. Yu et al. [9] performed a categorization of common coupling within kernel-based software and applied this categorization to the Linux kernel. They argued that without preventive actions, the maintainability of the Linux kernel would continue to be problematic. In a comparison of coupling in different OSS products, Yu et al. [10] found that the Linux kernel compared unfavorably with respect to three BSD kernels, namely, FreeBSD, NetBSD, and OpenBSD.

The above findings combined with our results show that OSS developers can take some preventive actions to improve quality. First, the quality assurance activities, such as inspections and testing, can be focused on highly coupled modules. Second, restructuring OSS software to reduce coupling can also improve quality in the long run.

6 Conclusion

The dynamic nature of OSS development requires a dynamic modeling approach to understand the relationship between coupling and defect proneness well. The traditional approaches that measure systems snapshots and count future defects cannot accommodate changing measures, added modules, deleted modules, etc.

Our modeling results showed that coupling has a significant effect on defect-proneness. Therefore, coupling should be monitored and managed in OSS projects to produce reliable and maintainable OSS products. The modeling approach explained here can be tightly integrated into an evolutionary OSS development in a seamless manner and can be used at any time while building models. The capabilities of the existing OSS tools can be easily combined for this purpose.

As the future work, we plan to collect data from additional OSS products and projects to generalize the identified relationship between coupling and defect proneness across a set of different OSS products.

References

1. Per Kragh Andersen, Ornulf Borgan, Richard D. Gill, and Niels Keiding. *Statistical Models Based on Counting Processes*. Springer-Verlag, 1993.
2. Lionel C. Briand, Jürgen Wüst, John W. Daly, and D. Victor Porter. Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems. *Journal of Systems and Software*, 51(3):245–273, 2000.
3. David R. Cox. Regression models and life tables. *Journal of the Royal Statistical Society*, 34:187–220, 1972.
4. Jr. David W. Hosmer and Stanley Lemeshow. *Applied Survival Analysis :Regression Modeling of Time to Event Data*. John Wiley & Sons, Inc., 1999.
5. Khaled El Emam, Saïda Benlarbi, Nishith Goel, and Shesh N. Rai. The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics. *IEEE Trans. on Software Engineering*, 27(7):630–650, July 2001.
6. Stephen R. Schach and Jefferson A. Offutt. On the Nonmaintainability of Open-Source Software. In *Meeting Challenges and Surviving Success: The 2nd Workshop on Open Source Software Engineering*, pages 52 – 54, Orlando, Florida, May 2002.
7. Katherine J. Stewart, David P. Darcy, and Sherae L. Daniel. Opportunities and challenges applying functional data analysis to the study of open source software evolution. *Statistical Science*, 21:167–178, 2006.
8. Terry M. Therneau and Patricia M. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer-Verlag, 2000.
9. Liguo Yu and Kai Chen. Categorization of common coupling and its application to the maintainability of the linux kernel. *IEEE Trans. on Software Engineering*, 30(10):694–706, 2004. Member-Stephen R. Schach and Member-Jeff Offutt.
10. Liguo Yu, Stephen R. Schach, Kai Chen, Gillian Z. Heller, and A. Jefferson Offutt. Maintainability of the kernels of open-source operating systems: A comparison of linux with freebsd, netbsd, and openbsd. *Journal of Systems and Software*, 79(6):807–815, 2006.

The Use of Open Source Software in Enterprise Distributed Computing Environments

A decision-making framework for OSS selection and planning

Jacob Krivoruchko
Nova Southeastern University, Fort Lauderdale, Florida, USA; University of Maryland University College, Adelphi, Maryland, USA; Advocate Health Care, Oak Brook, Illinois, USA. E-mail: krivoruc@nova.edu

Abstract. Firms increasingly rely on open source software for solving business problems and building mission-critical IT solutions. However, there are numerous issues associated with OSS, including its influence on the total cost of ownership (TCO) and supportability and upgradeability risks. While savings from obtaining a free copy of the software can be significant, software accounts for an average of 10% of TCO, while the majority of the costs are associated with project staffing. OSS requires significant investment into staffing because it needs to be carefully selected, customized, and installed. In addition, global communities may gather and dissolve at their will, so guarantees of support, revision, and bug fixes are minimal. Yet companies can gain competitive advantage through an ability to customize software to address specific business issues and exercising control over development, revision schedules, and modifications. OSS is not a panacea from the rising software costs. Instead, it is a serious initiative that has benefits, disadvantages, and risks associated with it.

Key words: Budget, business model, commercial software, distributed systems, enterprise computing, mission-critical project, open source software, reliability, risk, software selection, support, total cost of ownership.

1. Introduction

The open source market has evolved into a powerful force that is increasingly present in many areas of the industry, including web development, e-commerce, infrastructure management, financial applications, ERP, operations management, and more. Open source software (OSS) has established a strong presence among technology solutions that involve building client/server and complex distributed systems. Open source is not just a way to save money on the upfront software acquisition cost. It enables individuals and companies to tailor the base release to their needs, follow their own upgrade schedules (if needed), and coordinate development activities without any vendor involvement. While many early releases of OSS were rather unstable, the latest releases can effectively compete against the best products sold by giants like Microsoft, Sun, and Oracle, both in terms of functionality and security.

Many organizations become increasingly dependent on software vendors' release schedules, prices, support, and business models. Long project planning, development, and implementation cycles, along with complexity of the systems and dependency of the entire firm's operations, have made switching vendors a costly and undesirable process. Businesses struggle to cope with rising costs and products that do not fully address their needs. Some companies see a real opportunity to obtain a long-wanted freedom from their vendors' plans and ambitions through OSS, while others are cautious and prefer to avoid uncertainty of the unsupported products. Forrester's survey of 120 large North American companies shows impressive statistics: 46% of them already use OSS, while 14% have short-term plans to incorporate it into their existing computing environments. The same research has also revealed that European companies are not far behind: 31% of the surveyed 35 large firms use OSS, while only 17% do not have any plans for utilizing the free software code [3].

Is this a good time for your organization to acquire OSS and enjoy its benefits? Do not take advantages gained through the use of OSS for granted. In reality, OSS is not free. For companies it means dependence on the global community for further upgrades, problem resolution, and support; uncertainty about software stability and reliability; the need to hire and pay additional talent to modify and maintain software, along with other caveats [2]. The dilemma is whether OSS represents a real developer's dream and a panacea against rising software costs or a risky venture for companies mistakenly thinking OSS can help them save IT dollars.

2. Total Cost of Ownership (TCO) and OSS: Benefits vs. Risks

Many IT project managers consider only hardware, software, and infrastructure costs when budgeting for new initiatives. However, there are other critical components involved in the cost consideration. TCO refers to all costs incurred during system acquisition and full-cycle operation until its retirement. Acquisition costs include the processes of product selection, system design, purchasing, installation, deployment, and user training. The cost of system operation includes system management, maintenance, repair, user support, data center environment, and other factors that are highly specific to each individual environment [6].

TCO may also include other unexpected components, such as cost of poor performance, unexpected capacity considerations, satisfactory functionality, system availability, ease of user interface, and security. When these critical components fail or do not adhere to the end-user requirements, the cost of the project increases, thus having a direct influence on the TCO formula. It is not nearly enough to project all the costs associated with system acquisition and operation in order to accurately predict the total cost. The low price of an acquired system does not necessarily lead to cost savings, unless the system performs as expected, satisfies the users, and is available in accordance with service level agreements.

The price of software itself is low relative to TCO. It may represent about 10% of the TCO, with staffing costs adding a huge 50% to 70% portion [8]. This means that despite a common myth that software significantly reduces the overall available IT budget, its price has relatively low importance when compared to tasks following acquisition. This is where OSS may not stack up well against proprietary software. The myth of huge savings, therefore, loses its importance relative to the full-cycle development process. Moreover, TCO is difficult to calculate in the situation of uncertainty associated with lack of support and bug fix guarantee.

It is helpful to know what motivates companies to look for alternative software and doing away with traditional IT purchasing habits. The Emerald Hill Group - the company that manages pubs in Singapore - was able to customize its open source software, tailor it to its needs, and run on a reliable Linux distribution [10]. Gartner's research concludes that Windows XP costs on average 15% to 20% less to own and operate compared to Linux. But despite higher operational cost, the firm derived benefits from customization and running its own software under the company's control and on the firm's own revision schedule.

The Beaumont Health Care system in Ireland is a practical proof of how serious companies can be when it comes to OSS. The organization plans to save over 30 million euros over the period of five years by utilizing only OSS for desktop and front-office applications in addition to traditional infrastructure tools like Linux and Apache [5]. Considering the criticality of system availability in a clinical environment, this step signifies tribute to how far OSS has gone since its inception in terms of reliability and functionality. Beaumont and Emerald Hill Group also represent a new trend in OSS utilization - moving away from using separate software tools and modules and toward complete OSS environments that include more than just basic operating systems and web server applications. In other words, OSS makes its way into enterprise environments.

Many firms find it easier to build what they want out of OSS instead of living with unsatisfactory set of features provided by commercial suppliers. For example, highly individual knowledge management (KM) applications that depend on the users' needs are good candidates for OSS projects. As KM applications change and evolve together with an organization, frequent modifications are required in order to maintain an application up-to-date [4]. Such services may either be unavailable from a vendor or carry prohibitive costs. Education is another area where OSS is a highly desirable option. Low upfront acquisition cost, ability to avoid forced upgrades, flexible platform, and a chance to give students real tools to experiment with make OSS attractive to school district administrators [1]. However, given schools' usually tight IT operating budgets, the long-term costs might be too high, so it is important to weigh the benefits against ability to maintain the minimum level of staffing needed to keep the systems up and running.

There is a danger that firms and especially non-profit organizations may get too excited about low cost opportunities provided by OSS and forget about TCO considerations. This puts many organizations at risk of successfully implementing projects they cannot support. Firms considering switching to OSS environments from more traditional supported systems should also consider the risk of losing key personnel during critical project phases, expenses associated with migration, and retraining users and support people on the Linux or similar system, along with downtime required to accomplish the goals [7]. Smaller businesses lack both project management expertise needed to accurately calculate TCO and the budget needed to support their desired systems [9].

3. OSS Selection and Decision-Making Framework

The discussion about benefits and disadvantages of OSS can be summarized in two categories: TCO and freedom of choice for product development and maintenance. Some organizations implementing OSS will save money, others will be able to implement the exact kind of system they need to go about their business, and the rest of the firms will likely lose money or run into technical supportability issues [2]. IT managers must understand the savings model when it comes to budgeting technology projects.

While saving 10% to 15% of TCO on software acquisition is a significant achievement for any project manager, it is important to consider potentially higher costs of software selection, development, and maintenance. These costs are inevitable when it comes to OSS that must be modified, customized, compiled, and kept up-to-date. Those firms that employ a large number of IT staff and are in need of heavy customization for their business environments will likely reap the rewards of OSS, as they can save money on software licensing while going through customization efforts no matter what kind of software they own.

Many large firms do not want to accept responsibility for unexpected problems and prefer contracted software to the tools and applications coming from the community. Certainty may be worth the additional expenses in critical corporate business environments. This is just one of the reasons why commercial software's value will not diminish in the near future. It will still mark a sense of relative reliability, accountability, and certainty, assuming that the software vendor is financially stable to remain in business. However, we may also observe a rapid growth of OSS consumption. Some of it is associated with firms' desire to control their IT destiny - one of the best and better justifiable reasons to use OSS. Other reasons include savings, and many firms are on their way to disappointment in this case.

Based on this summary, we may derive the following general decision-making framework for software selection:

1. Clearly state and understand the project goals.
2. Discuss and fine-tune these goals with your end-users; finalize system requirements and ensure commitment to the project, including executive sponsorship.
3. Determine the criticality of the project, system availability and reliability requirements, and dependency of business operations on this particular software application.
4. Investigate whether there are any commercial software products offered on the market that will address the needs of the project. Solicit requests for proposal (RFP) and determine the initial costs.
5. If your project is long-term, critical, requires high system availability and reliable support, and its needs can be satisfied by commercial software, you may think about purchasing the product. OSS should still be considered if commercial product's cost is prohibitive, you expect future justified customization needs, or its features do not address the core project goals. In other words, ensure that all or any of the common factors pointing to possible OSS application are present.
6. If additional exploration is decided on, team up with senior architects and/or developers to run a small pilot project of OSS selection.
7. Determine if the selected software addresses the needs of the project, can rival or exceed commercial software's functionality and security, and represents initial financial savings.
8. Determine what tasks will be involved in the preparation of the selected OSS for final rollout and come up with a cost schedule for these tasks.
9. Compare the costs of development, customization, and installation between the OSS and commercial methods. Plug figures into the overall TCO formula, assuming you have also collected information about other components of the system. If these are not available, perform an analysis and make sure all other major components are present in the TCO calculation.
10. Determine the benefits of ownership and control over OSS relative to dependency on the vendor. The results will represent the intangible OSS project benefits.
11. Weigh the cost savings (if any) against the risks and determine the importance of intangible benefits discovered in the previous step.
12. Consider technical, supportability, operational, and financial risks associated with your software alternatives.
13. Make an educated final decision regarding software selection and architecture.

References

1. Alfonsi, B. (2005, June). Open source in the classroom. *IEEE Distributed Systems Online,* 3.
2. Berger, M. (2002, August 12). Eating the free lunch. *InfoWorld*, 24(32), 1.
3. Bruce, G.L., Wittgreffe, J.P., Potter, J.M.M., & Robson, P. (2005, July). The potential for open source software in telecommunications operational support systems. *BT Technology Journal*, 23(3), 79.
4. Donnelan. B., Fitzgerald, B., Lake, B., & Sturdy, J. (2005, November). Implementing an open source knowledge base. *IEEE Software*, 22(6), 92-95.
5. Fitzgerald, B. & Kenny, T. (2004, January/February). Developing an information systems infrastructure with open source software. *IEEE Software*, 21(1), 50-55.
6. Hawkins, M.W. (2001, June 22). Total cost of ownership: The driver for IT infrastructure management. *Prentice Hall PTR*; http://www.phptr.com/articles
7. Hutlestad, L. (2004, May 31). Linux TCO: Fact or fiction? *VAR Business*, 2012, 54.
8. MacCormack, A. (2003, August 18). The true costs of software. *Computerworld*, 37(33), 44.
9. Prichard, S. (2006, October 4). Why open source stays out of reach small and medium-sized enterprises. *Financial Times*, 6.
10. Vallejo-Yeo, G. (2005, May 14). Linux can save money. *Asia Computer Weekly*, 1.

Shared Assumption Concerning Technical Determination in Apache Web Server Developer Community

Juho Lindman
Helsinki School of Economics, Information Systems Science,
Runeberginkatu 22-24, 00101 Helsinki, juho.lindman@hse.fi,
WWW home page: http://www.hse.fi

Abstract. Our main finding is that OSS community seems to coordinate its activities by relying on technical determination. First, we review previous literature to understand OSS community coordination mechanisms. Then we empirically review OSS Apache Web Server community by using qualitative case study methods. Our data consist of developer list's email-discussions. Finally, we speculate that coordination rests on community's members' shared assumption concerning technical determination.

Keywords: Open Source Software, OSS, Coordination

1 Introduction

The Open Source Software (OSS) is a promising software development and distribution method. It is agreed that OSS can deliver business benefits [1]. There is, however, a gap in the previous research concerning on how community coordinates its efforts. If a company wants to adapt OSS processes, it has to understand what coordination mechanisms are used. Furthermore, those mechanisms are probably related to the community's shared understanding concerning OSS.

In this paper we outline mechanisms that are described in literature and pose question of how these mechanisms function in a software development community. Then we analyze a case community to find out can we find such mechanisms. Our research question is: is there a shared assumption concerning technical determination that enables coordination mechanisms?

2 Previous research

In OSS, there was originally a promoted understanding that if companies and communities would work together, everyone would be better off [2]. Recently, the relation between OSS communities and companies has been called into question. Dahlander&Magnusson [3] describe these kinds of relationships and divide them into 1) symbiotic, 2) commensalistic, and 3) parasitic depending on who is gaining from the relationship. If companies want to adapt OSS processes, they first need to understand how the communities function and how their coordination mechanisms are rooted in OSS cultural dynamics. That is only the first step in implementation, but the required organizational and financial changes fall outside of this study [4-5].

Coordination is required in software development to create consistent software [6]. This is especially true for OSS, since source code is easier to access. Ad hoc approaches might work for small projects, but more mature projects need ways to guarantee internal and external consistency of the software [7]. Traditionally coordination is defined as coordination activities toward common goal [7]. Malone and Crowston [9] redefine coordination as “managing dependencies between activities”. The latter seems to suit better for dispersed development activity, since it requires knowledge of dependencies, not knowledge of the activities [9].

Egyedi et al. [10] propose coordination mechanisms that coordinate the OSS development processes. They stress the importance of the 1) committee standardization, but focus more on: 2) bandwagon coordination, 3) regulatory coordination, 4) operational coordination, and 5) coordination by authority. Committee standardization means technical specifications for products in response to technical complexity [11]. Bandwagon mechanism means that popular OSS projects attract developers and set example. Regulatory mechanism means licenses, contracts and participatory agreements. Operational coordination mechanism means tools used in development. Coordination by authority means gatekeepers and controlling distribution contents.

Coordination mechanisms are built into OSS communities. OSS communities require high level of technical expertise [2] and build upon technical savvy "geek culture" [12]. An OSS development community can be viewed from organisation cultural perspective [13;15]. This culture consists of shared understanding how things are accomplished inside a certain developer community. OSS communities’ self-understanding is affected by influential writings of Eric Raymond [2] and Richard Stallman [15].

Raymond [2] described a motivational and coordinative entity of OSS: a plausible promise. Plausible promise is required to start OSS development. This promise will then motivate, but also direct the development activity. The promise includes not only goal, but also a process of how to get there. This correspondence between the end-product and the organisation, Conway’s law, was originally proposed nearly forty years ago [16]. The idea is that the software architecture mirrors the structure of the organisation that created it. Parnas [17] developed the idea further proposing that software modules should be regarded as job assignments not sub-programs. Ovaska [9] proposes that software architecture should be viewed as assignment communication tool used for coordinating interdependencies. This is in line with previous findings of Mockus [18]: interdependencies are controlled by limiting their number by using standard interfaces.

3 Method

We have outlined literature concerning coordination mechanisms and their relation to a technical subculture. In following sections we select one community and view how the coordination mechanisms function. Our research method is qualitative case study [19]. We do not seek statistical, but analytical generalizations. Therefore we need to focus only on one case community during quite limited time.

4 Data

We chose Case Apache Web Server (httpd) as our case community, because of its maturity. It has produced OSS code of quality comparable to the commercial actors of its domain and is currently the most used web server software. Apache has been researched before [10;18;20] since it is the first OSS "killer application". Originally the community was launched in 1995 based on the work of NSCA on httpd-program.

Our main data source is the Apache Web Server developer mailing-list (dev@httpd.apache.org) that can be found archived in the internet at http://mail-archives.apache.org/mod_mbox/httpd-dev. The data includes all emails (768) sent to the list between 1.1.2006-28.2.2006. We have also monitored other Apache foundations relevant email-lists that have something to do with httpd development. We have monitored the relevant Newsgroups archives and went trough the broad material offered in the portal web pages. The choice of data assumes that community's coordination happens on public email-lists. This choice seems justified as Apache promotes openness in its own action [21].

5 Case

We were interested in coordination mechanisms and their assumptions. We started out with Egyedi's [10] proposed five coordination mechanisms. Most of the emails discuss the technical specifications of software. The focus of this study is not about the functionalities and their quality, but on how coordination takes place. The discussion in the lists was very technical. Normally someone started a thread by posing a technical problem or question. Then the people on the list tried to tackle the issue and usually asked some additional information about the problem. Usually, a solution was offered after some discussion. Sometimes the question revealed that some additions needed to be made to the documentation or software. The overall development process seemed to follow the process proposed by Sharma [22].

From the five mechanisms most common was technical specification. This activity was not very formal, but followed problem-answer model. This mechanism was used more than all the other mechanisms combined. We also found signs of bandwagon mechanism, operational coordination mechanism, regulatory coordination and coordination by authority to function in case community.

Bandwagon mechanism was quite clear: the community self-image and popularity was reinforced in the official communication, but also in development email-discussion. The community was self-conscious and also able to attract interest and new members. Regulatory coordination was used. Higher involvement members were required to sign contracts. There were also positions of power, or hats [22] that could in some circumstances regulate action. We did not find in practice any clear instances of this kind of exercise of power in our data. The only seemingly accepted superiority was of technical nature. Operational coordination mechanisms were used. Concurrent Versioning Systems (CVS) coordinate activity as do status report emails that were automatically sent to the mailing list. Coordination mechanisms by authority were used. A release manager is chosen before major releases solve the open issues or to postpone them to the next version. Another authority coordination mechanism was that of gatekeepers. It takes a lot of technical skills to contribute to main modules of httpd. This gives those individuals that are capable authority over those that are not, based on meritocracy [22]. This authority is tied to individual, not organization.

We also found coordination mechanisms that do not fit very well to the proposed research framework. Those were: bug report control, voting and intentional consensus building. Bug fixing is a control mechanism, because it gives very important input and sets the agenda question to be answered. Bug reporting was made quite difficult because it required significant time and know-how even to submit a bug report. Furthermore, it seemed that given bug reports were not of very high priority [23]. This combined to the meritocracy will lead to a situation where outside input is quite low. Voting was another coordination mechanism. Voting can also be seen as a way to force decisions. In this case, it seemed that it was used rather as a way to test consensus. In the community, there was a very strong belief that voting should not be used as a coercive tool, but rather to see who has different opinion and why [24]. Third coordination mechanism that does not fall under previous categories was conscious community building. It seems that the active developer community is quite small. The rules of engagement seem to be written in the netiquette: rules of not only about form of communication, but also of content. At the same time, this mechanism enhanced coordination inside the email-dependant community by limiting the tone and content of messages. Another issue of community building was that the architecture was made in a way that the core httpd-module was not altered, but developments were rather redirected to more peripheral modules of the software. This creates consistency, but also limits the community size and offers possibility to re-allocate resources.

6 Results and discussion

Our research question was: Is there a shared assumption concerning technical determination that enables coordination mechanisms? All the five coordination mechanisms proposed by Egyedi [10] were used: committee standardisation, bandwagon coordination, operational coordination, regulatory coordination and coordination by authority. We also found three other possible coordination mechanisms: bug report control, voting and intentional consensus building.

All of these mechanisms seem to assume certain technical basis of how OSS communities function. We call this shared assumption technical determination. It means a belief that posed problems have technical solution, which will be found when the best developers focus on the problem - if not in the next version, then later. This also means that the software is never ready, but always developing.

This assumption enables the coordination mechanisms as follows: standardisation is required to reduce technical complexity by technical specifications Thus it is a technical solution. Bandwagon mechanism draws curious users and ambitious developers to successful communities. It rests on the belief in open source software is a "geek culture" of meritocratic technicians that are able to show their skills by developing software. Operational coordination mechanisms are technical solutions to problems related to learning, joint problem solving and division of work. Regulatory coordination is an agreement on how and which developer to credit for the contribution to the community. Coordination based on authority is required to smooth the releases of the continuously developing software and to make software available to larger population. Bug report coordination assumes that software is always developing and perceived difficulty of bug reporting hints that bug reports should be of high technical quality. Voting coordination tests consensus of the developers and strengthens the meritocracy by allowing merited programmers to vote. Community building coordination mechanism also strengthens the meritocracy and also assumes that the development effort of the software will continue for a long time.

7 Summary

We have outlined relevant literature of coordination, coordination mechanisms and organisational culture of a subgroup. Then we have selected a case community Apache Web Server and identified five mechanisms previously described in the literature and identified three other possible coordinating mechanisms. We have speculated that all these mechanisms rest on shared assumption of technical determination. These findings call for more research in the area of technical problem solving nature of OSS development.

References

1. B. Fitzgerald, The Transformation of Open Source Software, *MIS Quarterly,* 30 (4), 587-598 (2006).
2. E. Raymond, *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary* (O'Reilly Sebastopol, 1999).
3. L. Dahlander & M. Magnusson, Relationship between open source software companies and communities: Observations from Nordic firms, *Research Policy,* 34, 481-493 (2005).
4. J. Feller& B. Fitzgerald, *Understanding Open Source Software Development* (Addison Wesley, Boston, 2002).
5. M. Fink, *The Business and Economics of Linux and Open Source* (PHPTR, Prentice Hall, 2003).
6. E. Kraut & L. Streeter, Coordination in Software Development, *Communications of the ACM,* 38 (3), 69-81 (1995).
7. B. Curtis., H. Krasner, N. Iscoe, A Field Study of the software design process for large systems, *Communications of the ACM,* 31 (11), 1286-1287, (1988).
8. T.W. Malone, & K. Crowston, The interdisciplinary study of coordination, *Computing Surveys,* 26 (1), 87-119, (1994).
9. P. Ovaska, M. Rossi & P. Marttiin, Architecture as a coordination tool in multi-site software development, *Software Process: Improvement and Practice* , 8 (4), 233 – 247, (2003).
10. T. Egyedi & R.W. Van de Joode, Standardization and Other Coordination Mechanisms in OSS, *International Journal of IT Standards & Standardisation research*, 2(2), 1-17, (2004).
11. S.K. Schmidt & R. Werle, *Co-ordinating Technology. Studies in the International Standardization of Telecommunications* (MIT Press, Cambridge Mass, 1998).
12. R. Pavlicek, *Embracing Insanity: Open Source Software Development* (SAMS, Indianapolis, 2000).
13. J. Martin, *Organizational Culture: Mapping the Terrain*, (Sage, Thousand Oaks, 2002).
14. H. Schein, *Organizational culture and leadership* (San Francisco, Jossy-Bass, 1992).
15. S. Williams, *Free as in Freedom: Richard Stallman's Crusade for Free Software.* (O'Reilly, Sebastopol, 2002).
16. M.E. Conway, How do committees invent? *Datamation,* 14 (4), 28-31, (1968).
17. D. L. Parnas, On the criteria to be used in decomposing systems into modules, *Communications of the ACM,* 15 (12), 1053-1058, (1972).
18. A. Mockus, R. Fielding, & J. Herbsleb, Two Case Studies on Open Source Software Development: Apache and Mozilla, *ACM Transactions on Software Engineering and Methodology*, 11 (3), 309-346, (2002).
19. R. K. Yin, *Case Study Research, Design and Methods* 2nd ed., (Sage, Newbury Park, 1994).
20. R. Fielding, Shared leadership in the Apache project. *Communications of the ACM.* 42 (4), 42-43, (1999).
21. Apache Web Server, (2006a), http://www.apache.org/ foundation/how-it-works.html.
22. S. Sharma, V. Sugumaran, & B. Rajagopalan, A Framework for Creating Hybrid Open Source Software Communities, *Informations Systems Journal,* vol. 12 (1), pp. 7-25, (2002).
23. Apache Web Server, (2006b), http://issues.apache.org/bugwritinghelp.html
24. Apache Web Server, (2006c), http://www.apache.org/foundation/voting.html

Learning Through Practical Involvement in the OSS Ecosystem: Experiences from a Masters Assignment

Björn Lundell, Anna Persson and Brian Lings
University of Skövde, P.O. Box 408, SE-541 28 SKÖVDE, Sweden
{ bjorn.lundell, anna.persson, brian.lings }@his.se

Abstract. Increased awareness of and interest in Open Source has led to a number of university teaching initiatives, at both national and European level. In this paper we present experiences from a practical assignment designed to give students on an Open Source Masters course an insight into real involvement in Open Source projects. It discusses the motivations for the assignment, and how it was set up and executed. It reports on post facto student feedback, and reflects on a parallel, reduced exercise offered at undergraduate level. We find that the learning experience was both positive and valuable in that it gave real insight into Open Source participation, and also encouraged further participation in Open Source projects by students after the course had completed.

1 Introduction

This paper relates to a Masters course in Open Source and Distributed Development models. In particular, it focuses on the practical assignment for the course, which was specifically on participation in Open Source (OS) development. The course is a new addition to an existing Masters that has been running since 1990. The course was a natural outcome of the University's growing involvement in OS research, and in particular its partnership in two EU-funded research projects in the area (the FP6 project CALIBRE and the ITEA project COSI).

The course was motivated by the recent growth in interest and activity in OS nationally, in industry and the public sector. Apart from the recognised business opportunities offered by OS (Fitzgerald, 2006), there has been a growing awareness of the opportunities offered for organisational learning and individual skills development through participation in OS projects. Such awareness has led to a number of initiatives in teaching, particularly at the Masters level, at both national and European level (e.g. Fernandes and Machado, 2006; German, 2005; Megías et al., 2005; Özel et al., 2006).

However, the question naturally arises: can one teach about OS participation in a meaningful way in a classroom situation? There is a natural tension between the study of OS as a phenomenon, and OS as a way of contributing practically to community-driven projects. In designing the course this tension was recognised, and it was planned to at least partially address it through a practical assignment.

Please use the following format when citing this chapter:

Lundell, B., Persson, A. and Lings, B., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 289–294.

The aim of this paper is not to describe the complete course; instead the focus is on the practical assignment that the students carried out during the course.

2 The Masters Course

The graduate level course "Open Source and Distributed Development Models" (7.5 ECTS points) was offered at the University of Skövde during the autumn of 2006. The course was an international course, offered in English, and had 12 participants. To attend the course, a bachelor's degree in computer science or a closely related subject was required. The course aims to develop an understanding of open source software (OSS) systems and distributed development models for software systems and applications. Students taking the course are exposed to contemporary issues and industrial practice in OS, and the latest research results related to OS. Key to the course is student interaction with researchers and practitioners actively involved in the OS area, and real exposure to the Open Source ecosystem. The motivation for offering this course came from an observation of a growing student interest in OS and its relevance to industry, and was made possible by ongoing research activities in this area at the university.

Since 1990 the University of Skövde has adopted a profile for its advanced courses which has been strongly influenced by the UK model of postgraduate, research oriented Masters. It has a fairly conventional format, with class-room lectures from researchers and practitioners actively involved in the OS area (both from in-house speakers and invited speakers) and seminars based around research papers. The course has a number of intended learning outcomes (ILOs) against which students are assessed. In particular, in planning the course it was felt necessary that students should learn about the practicalities of what it means to contribute to an OS project. A practical assignment was seen as an effective way to expose students to real OSS ecosystems, and thereby contribute to two specific ILOs:

- Demonstrate knowledge of the major sources of information on OS projects and issues, and an ability to use these sources critically and effectively to report on OS projects and OS issues.
- Rigorously analyse work practices and development models used in Open Source and traditional Distributed Development projects.

The aim of the assignment is for students to get a feeling for how major open source style projects work in action, by contributing. All students had previous experience of using OSS. However, only one had experience of contributing to an OS project.

The assignment had two parts, both practical, which are described in the remainder of this section. In preparation for the assignment, two specific readings had been added to the reading list as a general background on open source projects and their philosophy. The first was "The Cathedral and the Bazaar" by Eric Raymond (Raymond, 1999) and the second the GNU Manifesto by Richard Stallman (Stallman, 2006).

Assignment Part 1: Introduction to Open Source Software in Practice

The first part was designed to exercise specific and limited aspects of contributing to OS projects, namely: (a) selecting an area to which the student felt it possible to contribute, (b) exposing a contribution to a wide community, and (c) learning tools/processes of a community.

Wikipedia (http://en.wikipedia.org) – a free, extensive, on-line encyclopedia entirely edited by its users – was the chosen medium for this exercise. More specifically, students were asked to make a contribution of content to Wikipedia. This had the advantage of giving a feel for open contribution without the overhead necessary to engage meaningfully with a software development community. Other advantages in choosing Wikipedia for the first part of the assignment include: (a) it is an OS project that a majority of the students are already familiar with; many of them use Wikipedia in searching for information on a regular basis, (b) it is easy to make a contribution as no special permissions or development tools are required, and (c) one does not need to wait for approval of a contribution, rather it is immediately visible. Only two requirements were expressed for the contribution: (1) it should include an English text of 150-200 words, and (2) it should include a picture.

Assignment Part 2: Open Source Development

The aim of the second part of the assignment was for students to gain an understanding of OSS development by taking part in a currently active OS project. We argue that a student's understanding of OS development will be considerably improved with practical experience of being part of a real project. It is hard to give insight into what an OS project really means unless it is grounded in experience.
Students were asked to make a contribution to one project (corresponding to about one week full time), and meanwhile examine the development work within the project. They could pick any project of their choice (Sourceforge and Tigris were recommended as starting points for exploration). A contribution could relate to code (new feature, bug fix, patch, etc), software testing, design specification, manual content, or web content.

Some interaction with the project was expected, and a genuine effort made by the student to make the contribution openly visible in the project. Ideally, a contribution would be accepted by the project and incorporated in the code/documentation base. However, it was recognised that in some projects this process may take longer than the course period and so was not a requirement.

In the end of the assignment period, students had to write an essay (2500 words plus appendices) of their contribution and the general development work within the chosen project, with a focus on the distributed perspective. In the text, they were required to position their experience with respect to the research literature. As a minimum, it was expected that each student would use at least two reviewed publications when relating their own observations and argument to the existing body of literature.

3 Experiences from the Assignment

This section describes student experience of the practical assignment as expressed in their final reports and a debriefing seminar which was held after completion of the assignment.

Assignment Part 1: Introduction to Open Source Software in Practice

All of the students used Wikipedia almost daily. Despite this none of them had made a contribution to the encyclopaedia before. The students found the material that Wikipedia provides about how to edit articles straightforward and rather easy to understand.

The students thought that the hardest part of the assignment was to find an article that they felt comfortable to contribute to. They perceived that the openness of Wikipedia encourages a lot of intelligent people to write on most topics, so most of the articles they looked into already had good coverage.

When finally finding an article, they put a lot of effort into making their text as good and correct as possible, knowing that many users read the website and that incorrect or badly written text will very likely be edited or even removed. They found it demanding to adopt the same style of writing as the rest of the article. Preparing for image upload held surprises. Firstly, no conflicting licenses can be incorporated into Wikipedia, so they were required to verify that the license on the uploaded material was appropriate for Wikipedia. Further, the whole upload section of Wikipedia has a flat hierarchy; all files are put in the same place in the structure, and referenced only by the filename entered for it when uploading.

In summary, students felt that contributing to Wikipedia was a positive experience and all of them claimed that they will make further contributions to Wikipedia in the future. They really appreciated the openness of the system which puts editing any content on the whole site only one click away.

Assignment 2: Open Source Development

Initially, some students thought that this assignment seemed very hard to carry out. The reason was that they assumed “contribute to an OS project” meant “contributing code”, which is a common misconception. On the contrary, there is actually a great need for contributions other than code – such as design documents, manual content, etc. When this was explained to the students they felt relieved as they lacked confidence in their own programming skills so were unsure of their ability to make a high-quality code contribution. However, five students did choose to contribute code, whereas the others decided to submit bug reports, documentation and desktop themes. One student wanted to concentrate only on submitting feature requests, but this was not considered appropriate as it lacked real engagement in the project. Examples of projects attracting student involvement include Premake, Robocode, KDE and Vim.

One thing that surprised some of the students was how well-organised their selected projects were. Their previous view of OS projects was of more or less chaotic anarchistic communities. What many students began to realise whilst undertaking the assignment was that many OS projects actually are quite well-structured with explicit work processes and clear roles.

The experience of most students was that their commitment to a project was well received. In the final report, one of the students wrote:

> “I don’t know if I should say that I was surprised by the kind treatment I got, but I was definitely surprised how glad they were to let me join the development team. The kind of welcome I got definitely made me more interested in joining open source projects if I find that something needs fixing.”

Wishing to continue involvement in OS projects was a recurring theme. For example, one student wrote:

> “I think all-in-all it was a good experience to make a real contribution to a real open source project. The open source phenomenon will surely stay for a long time to come and it is only positive to have gained a more direct understanding of how these work and function. In the future I think I will be much more inclined to get my feet wet in various open source projects.”

In a similar vein, another student wrote:

> “It is highly probable that I will contribute to other existing open source projects in the future, and it might also be the case that I will start one of my own to realize an idea that I have.”

One unanticipated problem occurred related to a student employed by a large company. The student got into trouble in making his contribution as the company he works for does not allow employees to make source code or any other contribution to any open source project without written authorization. Due to bureaucratic processes he had not received such authorization before the deadline for the assignment. However, he plans to submit his contribution to the project when such authorisation is received.

4 Discussion and Conclusions

Without any previous experience of the Open Source community, many students think that it is very hard to take part in Open Source projects. They think they have to be a well skilled programmer or technician to do it. It is important to change such perceptions.

One may speculate over the extent to which an assignment like this, in which students are forced to make a contribution, genuinely represents a learning experience with respect to Open Source participation. Might it contrast negatively with a regular voluntary contribution without any extra agenda? In mitigation, several points can be made. Firstly, the students chose the course from many options in the Masters programme; in fact it was amongst the most popular courses. Secondly, there was little restriction on the projects which could be chosen for making a contribution, and significant calendar time was made available for this. Thirdly, acceptance of contributions by the Open Source community was not a requirement for passing the assignment, only that a student must make a serious effort to contribute, supplemented by reported experiences and documentation of and reflections on genuine interaction with the project.

As we have experienced in this assignment, it may be a problem for people employed at a company to participate in a course like this. Some companies have an organisational policy in order to prevent making the company visible via Open Source contributions. However, it is clear that many companies value interaction with Open Source projects, and such courses are likely to attract company-based students. It is certainly worth our while to attempt to resolve such potential issues. It should also be noted that Scandinavia has a high degree of contributors to OS projects and the typically flat hierarchical structures in Scandinavian organisations may very well be an important background factor which reinforces this.

References

Fernandes, J. M. and Machado, R. J. (2006) A Two-Year Software Engineering M.Sc. Degree Designed Under the Bologna Declaration Principles, In *Proceedings of the International Conference on Software Engineering Advances (ICSEA'06)*, IEEE Computer Society, p. 1, 2006.

Fitzgerald, B. (2006) The Transformation of Open Source Software, *MISQ*, Vol. 30, No. 3, pp. 587-598.

German, D. (2005) Experiences teaching a graduate course in Open Source Software Engineering, In Scotto, M. and Succi, G. (Eds.) *Proceedings of the First International Conference on Open Source Systems*, Genova, Italy, 11-15 July 2005, pp. 326-328, <oss2005.case.unibz.it/Papers/OES/ES1.pdf>.

Megías, D., Serra, J. and Macau, R. (2005) An International Master Programme in Free Software in the European Higher Education Space, In Scotto, M. and Succi, G. (Eds.) *Proceedings of the First International Conference on Open Source Systems*, Genova, Italy, 11-15 July 2005, pp. 349-352, <oss2005.case.unibz.it/Papers/OES/EK3.pdf>.

Raymond, E.S. (1999) *The Cathedral & the Bazaar*, O'Reilly. hardcover ISBN 1-56592-724-9, October 1999.

Stallman, R. (2006) The GNU Manifesto, <www.gnu.org/gnu/manifesto.html> [Accessed 13 December, 2006].

Özel, B., Gencer, M. and Stephenson, C. (2006) An MSc Programme in Open Source Information Systems, In Özel, B., Burak Çilingir, C., and Erkan, K. (Eds.) *Towards Open Source Adoption: Educational, Public, Legal, and Usability Practices – OSS 2006 tOSSad workshop proceedings*, Como, Italy, June 10, 2006, TÜ_TAK (The Scientific & Technological Research Council of Turkey), TÜ_TAK MAM MAtbaasi, 41470, Kocaeli, Turkey.

Release Management in Free Software Projects: Practices and Problems

Martin Michlmayr, Francis Hunt, and David Probert

Centre for Technology Management
University of Cambridge
Cambridge, CB2 1RX, UK
martin@michlmayr.org

Abstract. Release management plays an important role in every software project since it is concerned with the delivery of a high quality product to end-users. This paper explores release practices employed by volunteer free software projects and shows problems that occur. A challenge that has been identified is the difficulty of coordinating a distributed team of volunteers in order to align their work for a release.

Key words: Release management, coordination, volunteers

1 Introduction

Release management is an important part of quality management since it is concerned with the delivery of high quality software to users [2]. Free and open source software (FOSS) is characterized by a highly iterative development model in which new development releases are typically made available very frequently. Despite the frequency of development releases in some projects, there are often problems with stable releases for end-users. The following two examples should illustrate this problem:

– Debian: in recent years, the project has faced increasingly delayed and unpredictable releases. Most notably, the release process of Debian 3.1 was characterized by major delays. Initially announced for December 1, 2003, the software was finally released in June 2005 – a delay of one and a half years. By the time the new version was released, the previous stable release was largely considered out of date and did not run on modern hardware.
– GNU tools: despite their popularity and importance, development has been slow in recent years and there is a long interval between releases. Version 1.13 of `tar` came out in August 1999, followed by version 1.14 at the end of 2004. The compression utility `gzip` saw a new version in December 2006, more than a decade after the last stable release in 1993. As a consequence of these long delays between stable releases, several vendors started shipping pre-releases.

Please use the following format when citing this chapter:

Michlmayr, M., Hunt, F. and Probert, D., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 295–300.

These examples show that release management is a real matter of concern. Despite the importance of this aspect of software production, little attention has been given to release management in FOSS projects [1].

This paper performs an exploratory study in order to get a better picture of actual practices and problems associated with release management in FOSS projects. The topic has been studied from a quality perspective and follows a similar approach to a previous paper which investigated quality practices and problems in FOSS projects [3].

2 Study

2.1 Methodology

For this study, interviews with twenty experienced developers from different FOSS projects were carried out. The interviews were conducted at a conference over the course of three days. Interviewees were either core developers or release managers of FOSS projects. The range of FOSS projects in which developers participated was very wide, ranging from small to very large and complex projects, and included projects of all types, such as desktop and server software and development tools. This great variety gives a good coverage of practices found in the FOSS community.

2.2 Types of Release Management

The study has revealed that the general term 'release management' is used to refer to three different types of releases. These types differ quite significantly regarding the audience they address and the effort required to deliver the release. The three types are:

– Development releases aimed at developers interested in working on the project or experienced users who need cutting edge technology.
– Major user releases based on a stabilized development tree. These releases deliver significant new features and functionality as well as bug fixes to end-users and are generally well tested.
– Minor releases as updates to existing user releases, for example to address security issues or critical defects.

Since developers are experts, development releases do not have to be polished and are therefore relatively easy to prepare. Minor updates to stable releases also require little work since they usually only consist of one or two fixes for critical or security bugs. On the other hand, a new major user release requires significant effort: the software needs to be thoroughly tested, documentation has to be written and the software needs to be packaged up. Since the main challenges are associated with the preparation of major new user releases, the focus will therefore be on releases aimed at end-users.

In terms of a project's release strategy, projects can be classified according to the following two release strategies:

– Feature based strategy: the basic premise of this strategy is to perform a new release when a specific set of criteria has been fulfilled and certain goals attained, most typically a number of features which developers perceive as important. This strategy is in line with traditional software development which is feature driven.
– Time based strategy: in this strategy a specific date is set for the release well in advance and a schedule created so people can plan accordingly. Prior to the release, there is a cut-off date on which all features are evaluated to decide whether they can be included in the release or have to be postponed.

2.3 Skills of the Release Manager

The role of the release manager is diverse and demanding because they have to interact with a large number of different people, understand technical issues but also know how to plan and coordinate. The following important skills have been identified:

– Community building: showing people that their input is useful. Release managers also need respect in the community in order to perform their work.
– Strong vision: showing developers in which direction the project should be moving.
– Discipline: saying 'no'. Release manager have to focus on overall goal and can not make everyone happy.
– Judgement: gauging the risk and possible impact of a particular change.
– Attention to detail: walking through every line of code that has changed.
– Good communication: writing release notes, asking for feedback, interacting with users.
– Management skills: talking to people, organizing, planning, making sure that different tasks are performed.

It is interesting to note that release managers in small and large projects play a vastly different role even though they essentially have the same responsibility, namely getting a high quality release out. In a small project, the release manager usually has an administrative role which involves the preparation of the release in different formats that can be distributed, the creation of release notes and the actual distribution of the software. In large projects, on the other hand, there is a big emphasis on coordination that needs to be performed by the release manager. They have to make sure that different parts of the software are ready at the same time and that all developers are aligned towards the common goal of creating a stable release.

2.4 Tools and Practices

Despite the important role release management plays in the delivery of quality software to users, there is little knowledge as to how FOSS projects perform releases. This section covers tools and practices employed during release management.

The study has revealed that there are few dedicated tools used in release management. However, many FOSS projects have tightly integrated their development tools with their whole development process, including release management. In particular, the use of version control and bug tracking systems serves important functions during release management as they give a good overview of the status of the project.

In addition to the use of tools, there are specific practices which are related to release management:

– Freezing: development is locked down in order to focus on the removal of defects and publication of the release.
– Scheduling: relatively few projects make use of a schedule but it is a vital component in those projects which employ a time based release strategy.
– Establishing milestones: some projects have loosely defined milestones but given the volunteer nature of most projects there is no guarantee that they will actually be achieved.
– Setting deadlines: many projects set deadlines but they are not always effective because the release manager has no control over volunteer participants.
– Building on different architectures: as part of a project's testing plan, it is beneficial to build the software on a number of different hardware platforms because each of them may exhibit bugs not visible on another platform.
– User testing: one of the main benefits from preparing a release is the feedback potentially obtained from a wide range of users. Even those projects which heavily deploy automatic test suites think that the most significant insights usually come from actual users. It is therefore important to make snapshots easily available.
– Following a release check list: a number of projects use a check list to make sure that all steps that are necessary to make a new release are followed.
– Holding a post-release review: surprisingly few projects have a formal post-release review but there are often informal discussions on the mailing list of the project, in particular when there were problems with the release.

2.5 Problems

As discussed earlier, the process of preparing a new stable release for end-users is quite elaborate and complex since the software needs to be sufficiently tested, documented and packaged for release. The release management process often faces certain problems, the most common of which are as follows:

- Major features not ready: planning in volunteer teams is very hard and it happens regularly that major features which are on the critical path are not ready. These blockers need to be resolved so the release process can continue.
- Interdependencies: with the growing complexity of software, there are increasing levels of interdependencies between software. For example, a piece of software may use libraries developed by another project or incorporate certain software components created elsewhere. This creates a dependency and can lead to problems with a project's release when those other components are not ready.
- Supporting old releases: there is generally a lack of resources in the FOSS community and in many projects old releases receive little support. Some vendors which distribute a given release may step in and offer basic support but it may still be problematic for other users of the software.
- Little interest in doing user releases: even though this study has emphasized the importance of user releases, many developers show little interest in making releases. Since developers by definition generally use the development version they often do not understand the need for a user release or do not see when a user release is massively out of date.
- Vendors shipping development releases: when projects do not publish new releases, some vendors start shipping development releases because they contain features or fixes which their customers need. This situation is problematic because it can lead to fragmentation and because development releases are generally not as well tested as user releases.
- Long periods without testing: some projects apply many changes with relatively little testing. At the time of the release, many issues are discovered and this leads to major delays.
- Problem of coordination: development in FOSS projects is done in a massively parallel way in which individual developers independently work on features they are interested in. Towards the release, all of this development needs to be aligned and these parallel streams have to stabilize at the same time. This can require substantial amounts of coordination.

3 Discussion

This study has shed light on practices and problems related to release management in FOSS projects, an important area which has so far been relatively unexplored. A major observation is that the preparation of user releases is associated with considerable problems, in particular in large projects. Small projects often face human resource problems but large projects deal with a more fundamental issue, namely that of coordinating a loosely defined team of virtual volunteers towards the common goal of making a release. These volunteers typically work independently in parallel development streams which require little coordination with other members of the project. However, during the preparation for the next release, these parallel streams need to be integrated and that requires coordination. Even more problematic is that each independent development stream has to be completed at the same time even though there is usually no schedule which provides guidance. When freezes are announced out of the blue, everyone wants to get their features and fixes in and the high number of changes pushes back the release date. Each delay is seen as a further opportunity to make more changes, thereby causing even more delays.

The present study has identified such problems in a number of projects but there is also evidence that some projects have found ways to deal with these problems. There is considerable interest among projects in the time based release strategy, in which time rather than features is used as orientation for the release and a schedule is followed. This release strategy is used in a growing number of projects, such as GNOME, OpenOffice.org and X.org. Time based releases promise solutions to cope with the complexity that occurs when a large team of distributed volunteers needs to be coordinated toward a common goal. However, further work is needed to study this release strategy in detail.

4 Conclusions

This paper has shed light on an important and as yet fairly unexplored area of FOSS development. The main finding of this study is that projects, in particular those with many developers, face severe challenges during the preparations for a release. These challenges are related to the coordination of a team which is not only large but also geographically dispersed and mainly consists of volunteer participants. While FOSS projects often rely on self-assignment of tasks with little coordination, all developers need to align their activities towards a common goal during release preparations. The time based release strategy has been suggested as a mechanism to coordinate large volunteer teams but further work is needed to explore this release strategy.

5 Acknowledgements

This work has been funded in part by Google and the EPSRC. Thanks to Fieldwave Ltd covering the expenses to attend the conference where the interviews for this study were conducted.

References

1. J. R. Erenkrantz. Release management within open source projects. In *3rd Workshop on Open Source Software Engineering*, pages 51–55. ICSE, 2003.
2. K. D. Levin and O. Yadid. Optimal release time of improved versions of software packages. *Information and Software Technology*, 32(1):65–70, 1990.
3. M. Michlmayr, F. Hunt, and D. Probert. Quality practices and problems in free software projects. In M. Scotto and G. Succi, editors, *Proceedings of the First International Conference on Open Source Systems*, pages 24–28, Genova, 2005.

CONTEXT-DEPENDENT EVALUATION METHODOLOGY FOR OPEN SOURCE SOFTWARE

Michele Cabano, Cesare Monti, and Giulio Piancastelli
DEIS, Dipartimento di Elettronica, Informatica e Sistemistica
ALMA MATER STUDIORUM — *Università di Bologna*
via Venezia 52, 47037 Cesena (FC), Italy
[michele.cabano, cesare.monti, giulio.piancastelli]@unibo.it

Abstract Many evaluation methodologies have been proposed to mitigate the risks of choosing Open Source Software as an effective solution to an enterprise's problem. This work extracts the shared traits from the most important and widely known evaluation models, and re-applies them to create a new methodology. This methodology has been designed both to be used for the creation of a common knowledge base, and to be specialized for application in the context of the particular breed of small- and medium-size enterprises found on the Italian ground.

1. Introduction

Selecting an Open Source Software (OSS) application as the appropriate solution to an enterprise's problem has always been an activity prone to many risks of different nature [4]. Proposed solutions to this choice problem have taken the form of evaluation methodologies [3, 1], aimed at assessing a software package's maturity for business IT adoption.

Open Source embracement is critical especially for small- and medium-size enterprises, where the consequences to a wrong decision in the picking of an Open Source alternative to a commercial product might not be as well-absorbed as in the context of bigger corporations. For this reason, the Italian region Emilia-Romagna funded the OITOS project,[1] started in 2005 with the goal of creating an Observatory for Innovation and Technological transfer on Open Source software. The project aims at strategic evaluations of Open Source solutions, in order to limit and control the risks for enterprise IT adoption. For that purpose, by re-applying a common structural pattern shared by the most important

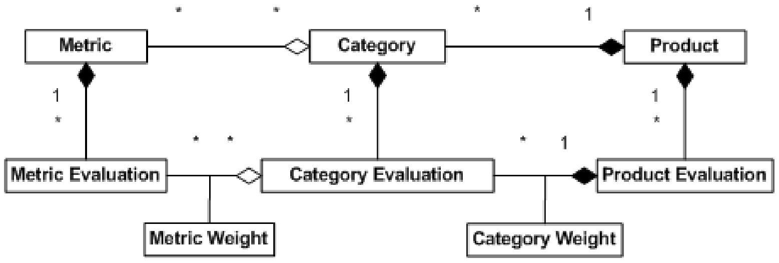


Figure 1. An UML diagram illustrating the evaluation meta-model, where a single metric can influence more than one category, and weights are used to take into account the final rating's dependency from the evaluation context.

and widely known methodologies, we designed a new OSS evaluation methodology, both to build a knowledge base classifying Open Source products, and to be applied in the specific business contexts of small- and medium-size enterprises participating in the OITOS project.

2. Modeling existing evaluation methodologies

We identified three most important works in the area of OSS evaluation methodologies, starting from the first Open Source Maturity Model [4], created by Bernard Golden at Navica; another Open Source Maturity Model has been later developed at the CapGemini consulting company [3]; finally, Intel Corporation, Carnagie Mellon University, and others are currently proposing a standard and open evaluation methodology for OSS, named Business Readiness Rating (BRR) [1].

Despite possible dissimilarities among the listed works, we believe each methodology always consists in the application of an *evaluation model* to one or more software packages during the execution of a well-defined *assessment process.* Both the process and the model can be reduced to a common pattern solving the Open Source evaluation problem.

2.1 The evaluation meta-model

As shown in Figure 1, all the examined evaluation models share a common meta-model subdivided along three layers. Those layers aim at refining a product's evaluation by aggregating raw data under ratings of similar assets, until an overall score is determined.

A metric is defined as a measurable property of an open source software project.[2] Metrics are organized into categories representing common areas of interest. The aggregation of categories represents the whole product under analysis. An evaluation is assigned to each metric, corresponding to the value reached by the software on that specific property.

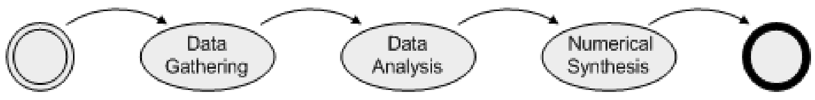


Figure 2. The three phases in the skeleton of the assessment process.

The evaluations of metrics sharing their category are combined into category evaluations, which are then aggregated into the final product rating. Weights could be assigned to categories and metrics to account for the context-dependent importance that some aspects of a product may have.

The examined models feature common categories to evaluate quantitative data taken from objective sources, such as the project's infrastructure. Differences between models result in the remaining categories, which try to capture subjective data and normalize qualitative information taken from other sources, such as code analysis or product use.

2.2 The assessment process skeleton

As illustrated in Figure 2, the prototypical *assessment process* for OSS evaluation methodologies is composed by three phases: (1) Data Gathering; (2) Data Analysis through predefined metrics grouped by category; (3) Numerical Synthesis. The process can be refined, as it happens with the Quick Assessment filter of the BRR [1], or further detailed, as in the seven steps of CapGemini's Maturity Model [3]; but the outlined three phase pattern still maintain its validity.

The Data Gathering phase can be accomplished by collecting information indirectly through reuse of third party evaluations, or directly from primary data sources such as the project's support infrastructure and web sites promoting the software product. The Data Analysis and Numerical Synthesis phases represent the application of the evaluation model in two steps: first, following a numeric scale, a score is assigned to each metric, then weighted and aggregated on the basis of metric's areas of influence to calculate the rating of each category; second, product's evaluation is obtained by applying different weights to category ratings.

The assessment process must be frequently repeated over time for each software package, due to the responsive nature of Open Source projects.

2.3 Towards a pattern for software evaluation

We believe that the meta-model and process skeleton detailed in Section 2.1 and 2.2 form the structure of a *pattern*[3] for the creation of the two key elements in a software evaluation methodology. Patterns distill and provide a means to reuse the knowledge gained by experienced practitioners [2]. In fact, the proposed software evaluation methodology pattern has been extracted from the solutions to the software evaluation problem proposed throughout the latest years. Besides, the creation of the BRR methodology shows that this solution pattern to the evaluation problem, on which the BRR is based, is considered mature enough to be openly discussed and worth of a standardization attempt.

3. The OITOS methodology

Started in 2005, the OITOS project is the first attempt made by the local government of an Italian region to spread Information Technology knowledge on its territory. The project aims at strategic evaluations of Open Source solutions under the perspective of enterprise IT adoption. The OITOS project involves a lot of companies which operate in several sectors. Different organizations have different technical requirements in the choice of software products: an assessment made for a company can rarely be totally reused for another. Thus, our primary task was finding a set of metrics which could always be used in any assessment, in order to create a knowledge base common to every organization.

3.1 Creating a knowledge base

Since existent evaluation models did not consider some important metrics the OITOS project's context demanded (*e.g.* migration costs or interoperability) we needed to create a new model to match our own requirements. The OITOS evaluation model was thus developed following the structure described in Section 2.1 without considering weight factors. The model was built to evaluate open source projects, rather than products, around three categories, aimed to measure a set of objective properties of general interest in a quantitative way: (1) Development; (2) Community; (3) Transition. The *Development* category measures the work made by project developers, and is composed by the Documentation, Developers, Tools, and Composability[4] metrics. The *Community* category measures all the resources offered by users, scientific communities and private companies, and is composed by the Visibility, Success Stories, Support, and Fundings metrics. The *Transition* category estimates how much the product is adaptable to existent organizational structures and configurable for its effective use, and is composed by the Configurability, Set-up, Use, Migration, and Interoperability metrics.

Needing general assessments in order to gain maximum reusability, we opted for the simplest possible evaluation process: it relied on a single phase called *Preliminary Assessment*, which included all the three steps highlighted in the prototypical process characterized in Section 2.2.

3.2 Context-dependent software evaluation

Companies participating in the OITOS project also needed evaluations customized to their particular technological context. Willing to reuse the objective and more general parts of each evaluation, the model and process described in Section 3.1 were extended piecemeal to produce a context-dependent methodology: metric weights from the meta-model in Section 2.1 were introduced, a new *Technology* category was added, and the process was extended to comprise three different phases.

Technology category groups all the metrics relative to features which an enterprise may consider important for its specific context. These metrics usually measure facets connected directly with the software product, never with the open source project's infrastructure. For this reason, rating those metrics is a subjective activity, but evaluations are more customized to their technological context than in other categories.

The assessment process is composed by three phases: (1) Context Analysis; (2) Preliminary Selection; (3) Filtered Selection. During the Context Analysis phase, information concerning a company are gathered in order to define its necessities. Each need is resolved by a software class which will be estimated by the model. After identification of software classes, a list containing possible products to introduce is edited, and a complete set of evaluation metrics is defined. In the Preliminary Selection phase, a set of most critical metrics is extracted. To each critical metric is associated a threshold, which must be surpassed to grant a software product access to the next phase. During the Filtered Selection step, software products passing Preliminary Selection are estimated once more, by the complete set of metrics defined in the Context Analysis phase. There are no thresholds assigned to the metrics: instead, each metric receives a 1 to 10 value which may be modified by a weight factor.

With reference to the process pattern outlined in Section 2.2, the Data Gathering phase has been split onto all the three phases in the OITOS methodology. During Context Analysis, data is collected about the company for which the evaluation is performed.[5] In the Preliminary Selection phase, we refined the process in the fashion of BRR's Quick Assessment filter [1]: data is gathered for critical metrics, and a first Data Analysis activity is performed to rule in or out software packages from the next step. During Filtered Selection, the complete evaluation of software is accomplished, featuring all the three phases in the pattern.

4. Open issues and future work

The Data Gathering and Data Analysis processes are driven by two sets of questions, which let evaluators collect the raw material to be used for assessing a product's rating, and establish the influence of each metric on the higher category layer. These two sets are of fundamental importance, since they comprise the point of view that the methodology enforces on the software product being evaluated. Unfortunately, even if those questions are sometimes contained in accompanying documents such as user's guides or appendixes, more often they are nowhere to be publicly found, especially when consultancy agencies are involved.

Despite the endeavor to control subjective information and normalize qualitative data, a certain degree of discretion is still possible while performing a software evaluation. Subjectivity has not been eliminated: it has just been transferred to the normalization process, which at most can be driven by a set of best practices, not a well-defined specification. If the influence of the evaluator's point of view and the dependence from the context cannot be eliminated, they should at least be traceable during and after the evaluation process. For this reason, every software evaluation should be accompanied by a document explaining choices and relating facts to the actual ratings assigned to metrics and categories.

In the context of the OITOS project, not only the results of every evaluation are public, but the methodology itself is made open to feedback from OITOS participants. As a first result, steady interest has been shown for the introduction of a new category in the model, with the purpose of evaluating legal issues connected with software, such as private data management, copyright, and licence responsibilities.

Notes

1. Details and documents for the OITOS project are available at `http://www.oitos.it`, including a case study on Groupware applications where our software evaluation methodology has been tested.

2. We adopted the BRR lexicon to achieve a uniform vocabulary and avoid confusion.

3. A pattern is a general repeatable solution to a commonly occurring problem.

4. Composability aims to estimate how many projects use the product as a component.

5. We extended the meaning of Data Gathering to also include information not related to software, but still essential to an effective evaluation.

References

[1] BRR Project (2005) Business Readiness Rating for Open Source. BRR-2005-RFC1

[2] Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M (1996) Pattern-Oriented Software Architecture – A System of Patterns. John Wiley & Sons

[3] Duijnhouwer FW, Widdows C (2003) Open Source Maturity Model. CapGemini Expert Letter

[4] Golden B (2004) Succeeding with Open Source. Addison-Wesley

Benefits and Drawbacks of Open Source Software: An Exploratory Study of Secondary Software Firms

Lorraine Morgan[1] and Patrick Finnegan[2]
1 University of Limerick, Ireland. Lorraine.Morgan@ul.ie
2 University College Cork, Ireland. P.Finnegan@ucc.ie

Abstract. Much of the assessment of OSS benefits and drawbacks has been based on anecdotal evidence appearing in practitioner publications, white papers, web articles etc. To a greater extent this research has tended to concentrate more on the technical benefits and drawbacks of OSS rather than their business counterparts. Furthermore, public administrations and companies operating within the primary software sector have traditionally been the focus for research on OSS benefits and drawbacks. Taking the viewpoint of IS/IT managers in 13 companies operating in the secondary software sector in Europe, this paper examines their experiences of the benefits/drawbacks of OSS.

Keywords: Open Source Software, Secondary Software Sector, Benefits, Drawbacks

1 Introduction and Research Motivation

The OSS movement has pragmatically shifted towards a more business-friendly and hybrid concept, and is now rapidly changing into a feasible alternative to proprietary software. Several innovative business models and new business opportunities have emerged as a result of the OSS phenomenon and many organisations have begun to capitalise on these [1]. Indeed, OSS plays a critical role in the business models for firms in high technology and other industries [2]. However, despite the considerable interest in OSS, there is a lack of published empirical research that rigorously examines the benefits and drawbacks of OSS. This is surprising considering there is an underlying assumption that the perceived benefits and drawbacks off OSS appear to be an underlying factor in its adoption. Our review identified the following benefits of OSS: reliability [3, 4]; security [3, 5]; quality [3, 6], performance [3], flexibility of use [4, 6]; large developer and tester base [6, 7]; low cost [8]; flexibility allowed by licenses [9]; user support from a community [6], escaping vendor lock-in [10]; increasing collaboration [1] and encouraging innovation [11, 12]. Our review also identified the following drawbacks: compatibility [13, 14]; security risks [15, 16]; installation problems [13]; lack of expertise [6]; version proliferation [6], user-friendliness [7]; lack of user support [17]; lack of ownership [7, 14]; insufficient marketing [6]; giving away the source code [18] and higher training investment in OSS [16].

Please use the following format when citing this chapter:

Nevertheless, given the dearth of extant research in this area, the benefits and drawbacks, particularly the business ones, relevant to OSS adoption are not well understood, as much of the research has been based on anecdotal evidence appearing in white papers [3, 4, 5, 16], practitioner papers [7] and web articles [10, 13, 14, 17, 18]. Furthermore, a great deal of this research has tended to focus mainly on public administrations and software companies operating within the primary software sector. This is rather surprising as Europe is the world leader in secondary development, a market that is rapidly taking the place of primary development [19].

Another important incentive for carrying out this research is the fact that this issue has not been addressed exclusively in the previous two Open Source Systems conferences held in 2005 and 2006. For instance, while the benefits of OSS were somewhat covered by Davini et al. [20]), this paper was more concerned with the use of OSS in the e-government area and did not address the drawbacks of OSS. Ven and Verelst [21] also presented a paper on organisational adoption of OS server software by five public administrations. Again, this study reported on five case studies in Belgian organisations currently using OS server software and focused more on the factors deemed important in the adoption decision. It is therefore argued that some rigorous analysis of the benefits and drawbacks of OSS experienced by managers operating in companies in the European secondary software sector would be timely.

2 Research Design

The objective of this study is to examine the benefits/drawbacks of OSS experienced by managers in firms in the European secondary software sector. The study was categorised as exploratory due to the scarcity of empirical work in this area. Thus, Marshall and Rossman [22] suggest that either a case study or field study research methodology can be used. The researchers decided that a field study would be appropriate as it would facilitate the collection of data from a larger number of organisations and would form the basis for more focused research at a later stage. Data collection was carried out using semi-structured interviewing in 13 companies (see Table 1).

Table 1. Companies Studied

Name	Informant
BSS Group PLC, UK	IT Contracts Manager
Combitech Systems,	Lead Engineer
Conecta, Italy	Head of R&D
Eircom Group PLC, Ireland	Technical Architecture Mgr
Eurocontrol Experimental Centre, France	Senior Researcher
Consult. Comp. (pseudonym), Switzerland	Consultant
Nokia Research Centre, Finland	Head of Software Technology
Phillips Medical Systems, The Netherlands	International Project Leader
Siemens AG, Germany	Program Manager
Sony Computer Entertainment Europe, UK	Linux for Playstation 2 Specialist
St. Galler Tagblatt AG, Switzerland	Chief Information Officer
Supertramp, UK	Technical Director
Vodafone, Spain	R&D Engineer, Head of R&D

Each interview lasted between forty-five minutes and two hours. Content analysis was undertaken using coding techniques proposed by Strauss and Corbin [23]. This approach seeks to develop theory systematically in an intimate relationship with the data, and can be utilised in the absence of, or in conjunction with, existing theory [23].

3 Findings

The ability to access the source code, modify it etc., has resulted in many of the technical benefits found in Table 2. However, it was found that many of the technical benefits, e.g. quality and the presence of a large developer and tester base only apply in some cases to more mature products like Linux, Apache etc. A new finding in the form of improved harmonization was also identified as another technical benefit. The business benefits outlined in Table 3 were seen as very significant for the interviewees, particularly escaping vendor lock-in, increased collaboration, and innovation. Although many of the benefits are similar to those found in the literature, some new findings also surfaced such as the extra business functionality experienced with OSS and establishment of de facto standards. In relation to the technical drawbacks of OSS, the findings from the study only support two of the technical drawbacks found in the literature (see Table 4), namely compatibility issues and lack of expertise. However, it was found that the lack of expertise issue tends to be more related to a lack of awareness about OSS. New findings in the form of poor documentation, proliferation of interfaces, less functionality and lack of roadmaps were considered chiefly to be the real drawbacks.

Table 2. Technical Benefits of OSS

Reliability	Reliability cited by majority as one of the main technical benefits in terms of high availability and dependability of applications
Security	Majority believed that OSS provides high security due to the availability of source code, the reduced threat of viruses and extra awareness of security in design phase of products. Two companies felt OSS would not be beneficial in terms of security
Quality	Majority of interviewees found quality beneficial in terms of enhanced quality from peer reviews and the quality of developers and testers. Two companies felt this could only be applied to top-tier, mature OSS products (e.g. Linux)
Performance	8 interviewees cited high performance in terms of capacity and speed. 3 have yet to see more evidence of how well OSS performs while 2 were uncertain if OSS performed any better than proprietary
Flexibility of Use	Beneficial for majority of interviewees because it facilitates changes, customisation, experimentations and allows freedom of choice
Developer & Tester Base	Very beneficial for majority as it ensures that OSS is quality software and is up-to-date.
Compatibility	Several mentioned that OSS is conducive to ensuring compatibility as it has a great interest in conserving formats for better interoperability. Remaining had not seen any evidence of this or it was not worth considering
Harmonisation	Improved harmonisation in interoperability and practices/operations

Table 3. Business Benefits of OSS

Low Cost	Half of the interviewees found this beneficial in terms of reduced licensing fees, upgrades, virus protection and the cost of the whole package, i.e. service and software. The other half considered low cost of no benefit
Flexibility by licenses	Seen by most as having a significant impact on reducing capital expenditure in company
Escapes vendor lock-in	Highly beneficial for most as it facilitates freedom of choice, gives sense of control and provides independence from private vendors. 2 companies felt vendor lock-in may also be experienced with OSS
Increases collaboration	Greater collaboration beneficial for majority as OSS facilitates product development, cooperation and exchange of knowledge, provides new ways of collaboration and permits sharing of expenses with other companies
Encourages innovation	Majority found that access to the source code facilitates more innovation; it produces ideas and encourages technical innovation while also creating more opportunities for innovation.
Extra business functionality	Beneficial because it results in ability to keep teams small which in turn improves productivity and communication
De facto standards	Not the only company doing something. Developing a standard that allows the company to focus on core competences would be beneficial

Table 4. Technical Drawbacks of OSS

Compatibility Issues	Not significantly disadvantageous but some companies experience compatibility problems with current technology, skills and tasks
Lack of Expertise	Some agreed that the average lay employee lacks expertise but this may be related to a lack of awareness of OSS
Poor documentation	Documentation outdated or may have died in development
Proliferation of Interfaces	Different builds often results in confusion in deciding which one to choose
Less Functionality	Level of integration not as good as Microsoft
Lack of Roadmaps	Makes it difficult for companies to see any strategic direction for vast majority of products. Most products don't have any strategic intent.

It was found that the business drawbacks outlined in Table 5 pose a bigger challenge for managers than their technical counterparts. For example, lack of support was considered a real drawback for the majority of the companies. Some of the companies have teams of technicians that can provide support internally. However, this is not always an option for many smaller organizations.

Table 5. Business Drawbacks of OSS

Lack of support	Majority felt that there was no safety net as there is no support and no company to back it up
Lack of ownership	11 found this a drawback as there is an inability to hold someone responsible or accountable for problems
Access to the source code	Several mentioned that others in the company may be uncomfortable with releasing source code. Lack of knowledge in relation to this issue
Insufficient marketing	Majority found this a drawback as no one organisation owns it all (OSS); there is no one to market it; OSS has no marketing budget which results in it being driven primarily by word of mouth
Investments for training	4 companies mentioned that training investments were higher for Linux than Windows. However, it was found that on e receives better quality in terms of training on OSS.
Finding the right staff/competencies	Can be difficult to find staff and develop competencies to work with OSS applications

4 Conclusion

This paper has built on extant practitioner-oriented examinations of OSS benefits and drawbacks by examining the technical and business benefits/drawbacks experienced by managers in companies in the European Secondary Software Sector. The ability to access the source code, modify it etc. has resulted in many of the technical benefits, e.g. reliability, security, flexibility of use and performance. It was also found that these benefits compared extremely well with proprietary software. The business benefits found in the study were just as significant for the interviewees and of equal value to them as the technical benefits, particular escape from vendor lock-in, increased collaboration and innovation. However, there was little support for findings from Krishnamurthy [6] that the *user support from a community* is quite beneficial to OSS because anyone using the software has an engaged community willing to answer questions. Only one of the companies found user support from the community to be a possible business benefit of OSS adoption. The remaining companies found user support from third parties, e.g. consultants, professional software houses more appealing.

The technical drawbacks identified by existing research e.g. version proliferation, security risks, installation problems, security risks, OSS being less user-friendly and troubleshooting and upgrading of OSS were not considered major drawbacks by the interviewees. In addition, there was no support for Kenwood's [7] assertion that OSS is less user-friendly, and few companies experienced installation problems. Finally, the business drawbacks found in the study depict a similar picture to those outlined in the existing literature. However, these drawbacks appeared to pose a bigger challenge for OSS than their technical counterparts.

References

1. Agerfalk, P.J., Deverell, A. Fitzgerald, B. and Morgan, L. (2005) Assessing the Role of Open Source Software in the European Secondary Software Sector: A Voice from Industry, *First International Conference on Open Source Systems*, Genova, 11-15 July 2005, 82-87.
2. Overby, E.M., Bharadwaj, A.S., Bharadwaj, S.G. (2006) An Investigation of Firm-Level Open Source Adoption: Theoretical and Practical Implications. In R.K. Jain (ed.), *Open Source Software in Business - Issues and Perspectives*, Hyderabad, India: ICFAI University Press, 2006.
3. Forge, S. (2006) *The rain forest and the rock garden: the economic impacts of open source software*, www.emeraldinsight.com/10.1108/146366906 10664633
4. Varian, H.R. and Shapiro, C. (2003) Linux Adoption in the Public Sector: An Economic Analysis. *Working Paper*, University of California, Berkley.
5. Coppola, C. and Neelley, E. (2004) *Open source – open learns: Why open source makes sense for education*, http://www.findarticles.com/p/articles/mi_qa4011/is_200401/ai_n9 466456/ pg_5

6. Krishnamurthy, S. (2003) A Managerial Overview of Open Source Software, *Business Horizons*, 46(5), September-October, 47-56.
7. Kenwood, C.A. (2001) *A Business Case Study of Open Source Software*, (http://www.mitre.org/ support/papers/tech_papers_01/kenwood_software/kenwood_software.pdf)
8. Open Source Initiative (OSI) (2006) *The Open Source Definition*, http://www.opensource.org/docs/definition.php
9. IDC Research (2005) Western European End-User Survey: 2005 Spending Priorities, Outsourcing, Open Source, and Impact of Compliance, http://www.itworldcanada. com/Pages/Docbase/ViewArticle.aspx?id=idgml-8f87ddb3-bfe 0-4b69&s=90323
10. Stafford, J. 2006: Time to plan your company's escape from Microsoft. http://searchopensource.techtarget.com/columnItem/0,294698,sid39_gci1163576,00.html
11. Wheeler, D. (2005) *Why Open Source Software/Free Sofware (OSS/FL, FLOSS, or FOSS)? Look at the Numbers!* http://www.dwheeler.com/ oss_fs_why.html.
12. Forrester Report (2000) *Open Source Cracks the Code*, www.forrester.com/ER/ Research/Report/ 0,1338,9851,00.html
13. Webb, M. (2001) *Going with Open Source Software: Is it the Right Choice for yourOrganisation?*http://www.techsoup.org/howto/articles/software/page1124.cfm.
14. Guth, S. (2006) Limiting factors for the adoption of open source software, http://claweb.cla.unipd. it/home/sguth/pdfs/OSS_gov_cons.pdf
15. Herbsleb, J. (2002) *Research Priorities in Open Source Software Development*, www. infonomics.nl/FLOSS/workshop/papers/herbsleb.htm
16. Forrester (2004) *The Costs and Risks of Open Source*, www.forrester.com/Research/ Document/ 0,7211,34146,00.html
17. Stevenson, J. (2005) Open Source Software: The Drawbacks, www.academon.com/lib/ essay/ open-source-software.html
18. Parsons, *A. Case Against Open Source*, http://www.builderau.com.au/program/ windows/soa/ Case_against_open_source/0,339024644,320265682,00.htm
19. Information Society Technologies Results (2005) *FLOSSing can make EU tech leader* http://istresults.cordis.europa.eu/index.cfm/section/news/tpl/article/BrowsingType/Featur es/ID/80607
20. Davini, E., Faggioni, E. and Tartari, D. Open Source Software in Public Administration. A real example OSS for e-government Observatories, *First International Conference on Open Source Systems*, Genova, 11-15 July 2005, 119-124.
21. Ven, K. and Verelst, J. (2006) The Organisational Adoption of Open Source Server Software by Belgian Organisations, *Proceedings of IFIP 8.2 Foundation on Open Source Software Conference*, 08-10 June 2006, Como, Italy, 111-122, Springer.
22. Marshall, C. and Rossman, G. (1989) *Designing Qualitative Research*, Sage Publications, California.
23. Strauss, A. and Corbin, J. (1990) *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Sage Publications, Newbury Park, CA.

Acknowledgement

This work is supported by LERO, the Irish Software Engineering Research Centre, University of Limerick.

Introducing Usability Practices to OSS: The Insiders' Experience

Stanisław Osiński[1] and Dawid Weiss[2]

[1] Poznan Supercomputing and Networking Center,
stanislaw.osinski@man.poznan.pl
[2] Institute of Computing Science, Poznan University of Technology,
dawid.weiss@cs.put.poznan.pl

Abstract. This paper presents a case study of introducing usability practices to a small open source project called Carrot^2. We describe our experiences from a point of view of an active Carrot^2 developer, who is at the same time a usability enthusiast and practitioner. We perform a critical analysis of the system's original user interface and describe the steps we took to improve it. We also analyse the success factors and the impact of the whole redesign process.

1 Introduction

The growing reliability of open source software (OSS) has led to widespread adoption of OSS server-side packages, such as the Apache web server or MySQL. At the same time, debates continued on whether OSS can also effectively replace proprietary end-user applications [1]. One important factor determining the success of the latter is the quality of their user interfaces (UIs). As opposed to traditional OSS users, who like taking on technical challenges, typical non-experts will simply want to complete their tasks quickly, minimising the effort required to learn and operate the software. Consequently, while the low quality of UI is unlikely to discourage experts from using OSS, it can potentially prevent massive adoption among end-users. Thus, problems of *usability*, i.e. the ease of learning and efficiency of using, of OSS are becoming increasingly important.

This paper is a case study of introducing usability practices to a small open source project called Carrot^2. We describe our experiences from a point of view of an active Carrot^2 developer, who is at the same time a usability enthusiast and practitioner.

2 Improving the user interface of Carrot^2

The aim of the Carrot^2 project is to provide a flexible framework for building *search results clustering* engines. Search results clustering is a web mining technique that simplifies browsing of search results by dynamically organising them into thematic folders (Fig. 1(d)). In response to the query 'tiger', for example,

Please use the following format when citing this chapter:

Osiński, S. and Weiss, D., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 313–318.

the user would be presented with search results divided into such topics as 'Mac OS', 'Tiger Woods', 'Security Tool' or 'Tiger Attack Helicopter'.

Carrot^2 started in 2002 with a primary goal to enable rapid experiments with novel web mining techniques [2]. But as the project matured, we decided to place greater emphasis on supporting end-users, which required making Carrot^2's web-based interface easier to learn and use. We divided the process of redesigning the UI into four major stages: evaluation of the original design, prototyping of the new design, usability testing, implementation and evaluation of the redesigned version, all of which we describe below.

2.1 Evaluation of the original design

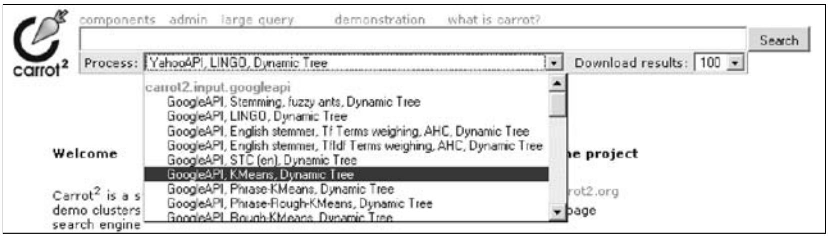
The original design of the user interface was aligned with the system's primary goal at that time – attract text mining researchers – and it perfectly served its purpose. After shifting the emphasis to end-users, however, some elements of the UI would not be appropriate anymore. Below we summarise the most important problems that we planned to address.

Choosing sources of search results. The original UI combined two different aspects – choosing the source of search results and the clustering algorithm – into one *Process* combo-box (Figure 1(a)). One problem with this approach was that these two choices are independent, and the original UI made it impossible to use certain combinations of search sources and clustering algorithms. A more serious problem, however, was that many end-users would care less about changing the clustering algorithm than the search source. Unfortunately, the original UI would force them to choose e.g. between: *'GoogleAPI, English stemmer, Tf Terms weighing, AHC, Dynamic Tree'* and *'YahooAPI, LINGO, Dynamic Tree'*, both of which are meaningless for a non-expert.

Choosing search results sorting order. The presentation of search results and thematic folders in the original UI was based on a variant of the two-panel selector pattern, in which the folders were shown on the left-hand side of the screen and the corresponding search results appeared on the right (Fig. 1(b)). One element of this interface that might be confusing to non-expert users was the choice of results sorting method located above the folder list. Not only was it put in the inappropriate visual context (sorting order referred to search results shown on the right and not to the folder list), but also it was not clear what different sorting orders meant and which of them was currently selected.

Technical jargon. At a few places the original UI used technical and specialised vocabulary not easily understood by non-experts. One example is labeling the combo box for choosing the search results source (Yahoo!, Google, MSN) as *Process*[1]. The combo box itself contained hard to understand items such as: *'Carrot^2.input.snippet-reader.alltheweb'* (internal identifier of a search results source) or *'Rough-KMeans'* (name of a text clustering technique).

[1] The internal architecture of Carrot^2 is based on pipelines called *processes*.

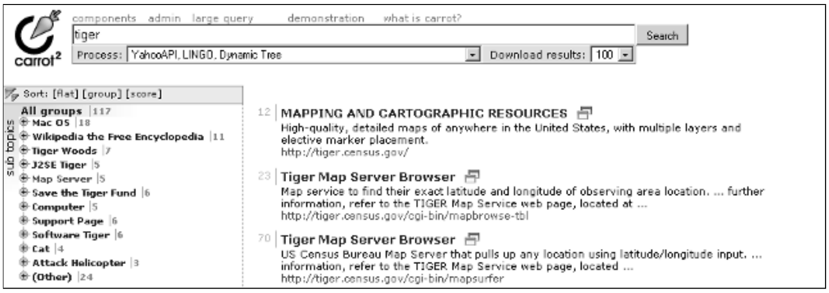


Process: YahooAPI, LINGO, Dynamic Tree
Download results: 100

carrot2.input.googleapi
GoogleAPI, Stemming, fuzzy ants, Dynamic Tree
GoogleAPI, LINGO, Dynamic Tree
GoogleAPI, English stemmer, Tf Terms weighing, AHC, Dynamic Tree
GoogleAPI, English stemmer, Tfidf Terms weighing, AHC, Dynamic Tree
GoogleAPI, STC (en), Dynamic Tree
GoogleAPI, KMeans, Dynamic Tree
GoogleAPI, Phrase-KMeans, Dynamic Tree
GoogleAPI, Phrase-Rough-KMeans, Dynamic Tree
GoogleAPI, Rough-KMeans, Dynamic Tree

Carrot² is a s
demo clusters
search engine

(a) Startup screen of the original user interface.


components admin large query demonstration what is carrot?
Search
carrot²
Sort: [flat] [group] [score]
sub topics

tiger
Process: YahooAPI, LINGO, Dynamic Tree
Download results: 100

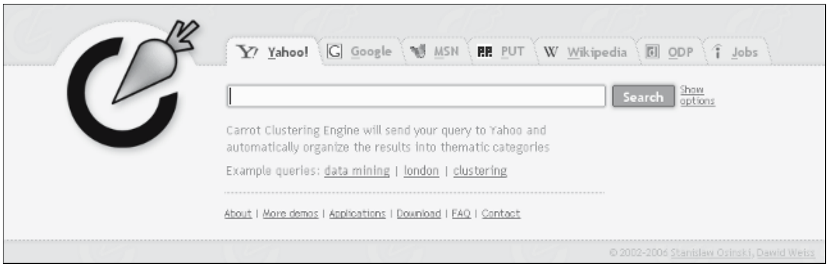
All groups 117
Mac OS 18
Wikipedia the Free Encyclopedia 11
Tiger Woods 7
J2SE Tiger 5
Map Server 5
Save the Tiger Fund 6
Computer 5
Support Page 6
Software Tiger 6
Cat 4
Attack Helicopter 3
(Other) 24

12 MAPPING AND CARTOGRAPHIC RESOURCES
High-quality, detailed maps of anywhere in the United States, with multiple layers and elective marker placement.
http://tiger.census.gov/

23 Tiger Map Server Browser
Map service to find their exact latitude and longitude of observing area location. ... further information, refer to the TIGER Map Service web page, located at ...
http://tiger.census.gov/cgi-bin/mapbrowse-tbl

70 Tiger Map Server Browser
US Census Bureau Map Server that pulls up any location using latitude/longitude input. ... information, refer to the TIGER Map Service web page, located ...
http://tiger.census.gov/cgi-bin/mapsurfer

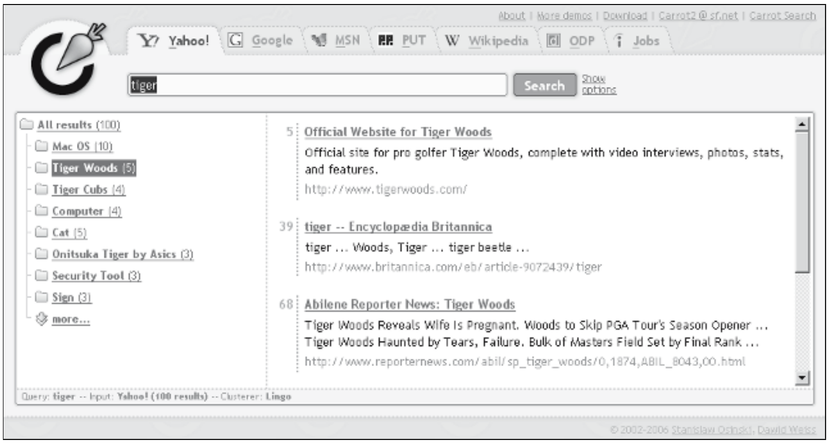
(b) Results page of the original user interface.


Yahoo!
Google
MSN
PUT
Wikipedia
ODP
Jobs
Search
Show options
About | More demos | Applications | Download | FAQ | Contact
© 2002-2006 Stanisław Osiński, Dawid Weiss

Carrot Clustering Engine will send your query to Yahoo and automatically organize the results into thematic categories

Example queries: data mining | london | clustering

(c) Startup page of the new design.


About | More demos | Download | Carrot2 @ sf.net | Carrot Search
Yahoo!
Google
MSN
PUT
Wikipedia
ODP
Jobs
Search
Show options
© 2002-2006 Stanisław Osiński, Dawid Weiss

tiger

All results (100)
Mac OS (10)
Tiger Woods (5)
Tiger Cubs (4)
Computer (4)
Cat (5)
Onitsuka Tiger by Asics (3)
Security Tool (3)
Sign (3)
more...

5 Official Website for Tiger Woods
Official site for pro golfer Tiger Woods, complete with video interviews, photos, stats, and features.
http://www.tigerwoods.com/

39 tiger -- Encyclopædia Britannica
tiger ... Woods, Tiger ... tiger beetle ...
http://www.britannica.com/eb/article-9072439/tiger

68 Abilene Reporter News: Tiger Woods
Tiger Woods Reveals Wife Is Pregnant. Woods to Skip PGA Tour's Season Opener ... Tiger Woods Haunted by Tears, Failure. Bulk of Masters Field Set by Final Rank ...
http://www.reporternews.com/abil/sp_tiger_woods/0,1874,ABIL_8043,00.html

Query: tiger -- Input: Yahoo! (100 results) -- Clusterer: Lingo

(d) Results page of the new design.

Fig. 1. Carrot^2 user interface screenshots, available on-line at http://www.carrot2.org.

The good points. Despite the deficiencies highlighted above, the original design had many elements worth preserving in the improved UI. The most important one was the 'stateless operation' paradigm – all search results and cluster folders should be fetched in one request. With this approach the users would not experience any delays caused by additional communication with the server while browsing the folder list.

2.2 Prototyping of the new design

To allow ourselves a large degree of flexibility while brainstorming and polishing the new UI, we decided to use a prototyping technique based on static HTML files. This approach let us keep the cost of changing the user interface to a minimum and perform usability testing before writing any code. Over a few weeks, we worked on the prototype in an iterative fashion, increasing its fidelity and fixing problems identified by usability tests. In the last step, we ensured that the prototype displayed correctly on the major web browsers. Below we highlight the most important changes we introduced to the Carrot^2 user interface.

Search results sources as tabs. To address the most important problem with the previous UI, we decided that different data sources should be represented as 'tabs' (Fig. 1(c)), like in conventional search engines. The choice of a clustering algorithm, previously integrated with the search source, was delegated to a separate UI component. With this approach users can try all possible combinations of data sources and clustering algorithms.

Hidden clustering algorithm choice. As the majority of end-users will not have enough technical knowledge to purposefully select the clustering algorithm, this choice is only available in the block of 'advanced options', shown after selecting the *Show options* link placed next to the query field.

Polished startup and results pages. The redesigned user interface offers an enhanced startup page with a one-sentence description of what Carrot^2 will do with the query typed in by the user. The search results page, apart from the introduction of search source tabs, has not undergone any major paradigm changes. We preserved the previous layout (Fig. 1(d)) and stateless behaviour of the original UI. Minor improvements included: enabling independent scrolling of search results and folders by putting them in separate internal frames, making folder selection easier by slightly increasing the font size and spacing, adding icons to strengthen the 'foldering' metaphor. The the choice of search results sorting method was removed entirely, and the results are always shown in the order they were received from the search engine.

2.3 Usability testing of the new design

Following the experiences of Jacob Nielsen [3], who showed that testing a UI even with a handful of people can detect the most important design flaws, we performed a number of informal usability tests of the new UI. For each test participant, we briefly introduced the Carrot^2 clustering engine and explained the goal of the tests, emphasizing that it is the UI that is being evaluated and not the participant's actions. Then, we asked to perform a number of search tasks using the UI, e.g. *query Google for 'data mining' and find results related to the field of text mining.* Throughout the tests, we did not help the participants to complete their tasks, but only carefully observed their actions and took notes of their interaction with the system.

The most important observation we made during the usability tests was that for some specialised data sources the users did not know what type of queries they could submit. To give such users some guidance, on the startup page we included example queries for each search tab.

2.4 Implementation and evaluation

An important input for the implementation phase was the static HTML prototype of the UI. With a high-fidelity prototype covering all the details of the UI, the implementation work could concentrate on providing an efficient technical backbone for the already designed graphics and interaction schemes.

A rather imperfect (but valuable) test of the impact of the new design is comparing the number of queries the public demonstration of Carrot^2 handled before and after the UI change. Before the first beta release of the improved user interface, which happened in August 2006, the monthly number of queries did not exceed 5 000. The query rate soared to over 13 000 in August 2006 and it further increased in September, when the new release of Carrot^2 was officially announced. Although the numbers at the end of 2006 may still to some extent be a result of the new release advertising, the popularity of Carrot^2 on-line demo seems to have stabilised at a much higher level. In addition to pure numbers, we also received quite a few favourable comments from users who liked the improved concepts and graphical finish of the new UI, which is a very important motivating factor for future work on the project.

The whole process of user interface redesign took a total of about 200 working hours spent between March and September of 2006. The two largest work items involved were building a prototype of the new UI and its actual implementation. The former took about 80 working hours spent between March and June of 2006, while the latter required about 120 working hours in the period between July and September of 2006. Compared to the total effort put into Carrot^2 since 2002, the UI-related work does not seem very costly.

3 The success factors of usability practices in Carrot^2

While open source developers declare that usability has a high or very high priority for them, they rarely apply it in practice with respect to their own projects [4]. One reason for this is, as explained in [5], that *most work on open source projects is voluntary, developers work on the topics that interest them and this may well not include features for novice users* and usability. Below we summarise the factors that we found important for a successful introduction of usability practices to Carrot^2.

Usability enthusiast among project developers. One of Carrot^2's developers, in addition to being interested in web and text mining problems, is a usability enthusiast. This made the voluntary work on user interface aspects both interesting and rewarding. Moreover, by combining the role of a developer knowing Carrot^2's internals and a usability expert, we could avoid, for example, designing a UI that is impossible to implement due to technical reasons.

*Usability work was part of a major system architecture change.*The user interface redesign was carried out as part of a major refactoring of the system core. As a result of the change, we would have to make a large number of changes to the implementation of the original UI. This gave us a good reason to completely abandon the original UI and its implementation and create the new one from scratch. Unfortunately, throwing away large parts of code is a situation which many open source projects, especially those with public programming interfaces (APIs), simply cannot afford.

Very simple user interface. The new Carrot^2's UI we designed contained only two screens, which made it relatively easy to pay attention to every detail. For projects with larger UIs or a large number of distributed contributors, it may not be possible to carry out the usability work in a similar way.

4 Conclusions and future work

In this paper we described the process of introducing usability practices to a small OS project called Carrot^2. A number of favourable factors, such as having a project member who shared the developer and usability expert roles, contributed to the success of the UI improvement process. One lesson we learned during our work on usability is that creating a high-quality user interface does not necessarily have to cost much and can bring substantial benefits.

References

1. Relevantive: Linux Usability Study Report. On-line: http://www.linux-usability.de/download/linux_usability_report_en.pdf (2003)
2. Weiss, D.: Carrot^2: Design of a Flexible and Efficient Web Information Retrieval Framework. In: Proceedings of the Third International Atlantic Web Intelligence Conference, AWIC-2005, Łódź, Poland. Volume 3528 of Lecture Notes in Computer Science., Springer (2005) 439–444
3. Nielsen, J.: Why You Only Need to Test With 5 Users. On-line: http://www.useit.com/alertbox/20000319.html (2003)
4. Andreasen, M.S., Nielsen, H.V., Schroder, S.O., Stage, J.: Usability in Open Source Software Development: Opinions and Practice. Information Technology and Control **35A**(3) (2006) 303–312
5. Nichols, D.M., Twindale, M.B.: The Usability of Open Source Software. First Monday, On-line: http://www.firstmonday.org/issues/issue8_1/nichols/ **8**(1) (2003)

Perceptions on F/OSS Adoption

Bulent Ozel[1], Uros Jovanovic[2], Beyza Oba[3], and Manon van Leeuwen[4]
1 Carnegie Mellon University, Pittsburgh, USA, bulento@cmu.edu
2 XLAB, Ljubljana, Slovenia, uros.jovanovic@xlab.si
3 Istanbul Bilgi University, Istanbul, Turkey, boba@bilgi.edu.tr
4 FUNDECYT, Badajoz, Spain, manon@fundecyt.es

Abstract. This paper aims to reveal results of a survey run by the tOSSad[1] project. The majority of survey variables devised to capture perception of public administrators around Europe regarding the importance they attach to the factors such as F/OSS product quality, availability of support, expertise and documentation, TCO, vendor lock-in, political influence, administrative attitudes, productivity, and training costs, all of which intermingle with financial, technical, legal, and personal issues. The analysis consist of depiction of respondents' administration profile in terms of their F/OSS usage and adoption, descriptive summary and analyses of factors mentioned above, and statistical inferential analyses of survey items. Some valid statistical tests are conducted to understand, to discuss and to see the extend and significance of any F/OSS adoption generalizations for Europe based on the findings of this particular survey.

1 Introduction

Survey respondents are selected from public bodies at middle to high level of managerial positions. A prior knowledge or experience on F/OSS of administrators has been an implicit factor in selecting them as respondents for the survey. However, respondents' F/OSS knowledge depth, their experience level and type, software acquisition policy of their administration, their administrative or personal attitude towards F/OSS, etc have been disregarded at the phase of convenience sampling. The survey covers 13 different countries around Europe with a slightly varying number of respondents for each of them. hey are Ukraine, Turkey, Sweden, Spain, Slovenia, Norway, Malta, Italy, Germany, Bulgaria, Austria, France, and Netherlands. The number and regional diversity of our sampling allows us to run some key descriptive and inferential statistical methods.

2 Analysis

The analyses we have run consist of depiction of respondents' administration profile in terms of their F/OSS usage and adoption, descriptive summary and analyses of

[1] tOSSad is an EU project funded under FP6 IST program, www.tossad.org.

factors mentioned above, and statistical inferential analyses of survey items. Some valid statistical tests are conducted to understand, to discuss and to see the extent and significance of any F/OSS adoption generalizations for Europe based on the findings of this particular survey. Most of the survey items are five-point Likert scaled [1, 2]. All figures, statistical summaries and tests are generated or devised manually using R Project2, a very detailed analysis and discussion of this survey is available at the tOSSad project Web portal.

2.1 Descriptive Results

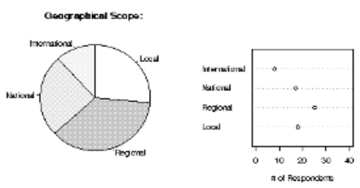


Fig. 1: Geographical scope of respondents' activities

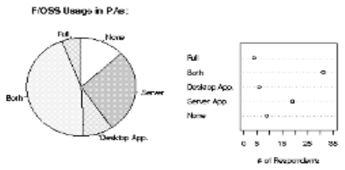


Fig. 2: F/OSS usage at public institutes of respondent PAs

The vast majority of the survey respondents are working in public organizations which have either national or regional level activities. Although a knowledge and acquaintance on F/OSS has been a loose precondition in our sampling, 9 of them, out of 69 valid respondents, declared that they do not use any F/OSS product within their organization or it is not clear to their knowledge. Along with F/OSS usage at any level, we have asked whether our respondents are actively participating, collaborating, or supporting any F/OSS related projects. It is seen that about %77 of the survey respondents have taken part in F/OSS related projects. We have seen that F/OSS project participation of public administrations parallels its adoption within their organization which is an inherently expected picture.

With the first group of survey items we have attempted to understand the importance of quality, support, and documentation in views of the respondents towards F/OSS adoption at their public institutes. A summary of responses are given on Figures 3, 4, and 5. Apparently, we can conclude that F/OSS product quality alone has a direct impact on its adoption. World wide observations elsewhere[3] also support that fact, for instance, increase of F/OSS usage on desktops entangles with its progress in becoming more and more end user friendly. Importance attached to

[2] http://www.r-project.org/

accessibility of expertise and support, availability of documentation have a rather bell shape distributions which imply that they are not a significant factor as a F/OSS adoption barrier. This picture might be an important input to the debate about documentation and manuals necessary for F/OSS migration in Public Administrations. Considering the fact that respondents of this survey have an experience or knowledge on F/OSS and an the fact that a major group of them have deployed or have been involved in F/OSS deployment within their administration we can say that our study rather supports the view that availability of books, manuals and other documentation is not an important barrier factor against F/OSS adoption in public bodies.

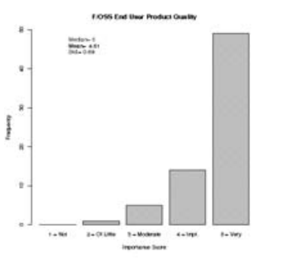
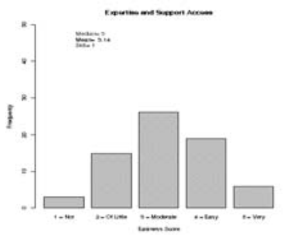
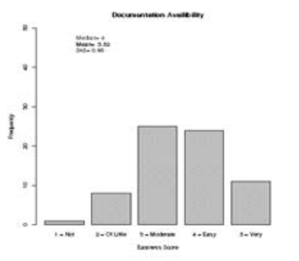
		

Fig. 3: Quality	**Fig. 4:** Customer support and reachability of experts	**Fig. 5:** Availability of F/OSS product documentation

A group of survey items have been placed within the survey to examine effect of built-in barriers such as political decisions, administrative and personal attitudes towards F/OSS in general. This finding shows that along with technological, service, and acquirement costs 'political influence' is also a determinant factor in F/OSS deployment within public administrations. Furthermore, it is interesting to notice that even though responses to other technological and financial factors we have examined so far show a moderate impact on F/OSS adoption, political influence as a factor in the issue has a relatively sharper implication of being a barrier.

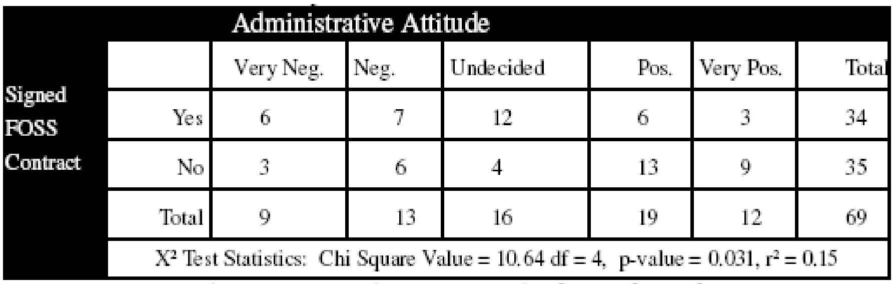
Respondents are asked to rate first their attitudes then attitude of their administration about F/OSS. We have seen that the positive personal attitude of the respondents migrate to relatively more negative levels when they are asked to rate general attitude of their administration about F/OSS. We can interpret about this discrepancy in several ways. The first explanation might drive upon the fact that F/OSS advocacy or acceptance is rather more common at personal level than at institutional level. In fact, diffusion of innovations which is theorized and formalized by Everett Rogers[4] supports the observation of our survey which is in line with that fact. When we assume that F/OSS as a whole is an innovative technology, then according to diffusion of innovations theory educated individuals with multiple info sources have greater propensity to take risk in evaluating, trying, and adopting new technologies. A second explanation of such a discrepancy might partially stem from the unavoidable bias that might occur in a survey study [2]. It is highly probable that we have been responded by individuals with relatively more affinities to F/OSS, that is, by those having higher willingness and ability to adopt F/OSS depending on their awareness and interest.

The respondents are asked whether they have faced any problem while introducing F/OSS at their administration. Those with negative answers are further asked to explain the reason(s) of their bad experience(s). Textual analysis of their answers reveals four basic sources of bad experience at the introduction phase of F/OSS deployment in public administrations. The first one, the most common as well, is the resistance to change. The second one is the lack of visions on F/OSS business model. Third one is its unsatisfactory compatibility with proprietary file formats. The last reported source of bad experience is lack of satisfactory GUI desktop distributions.

2.2 Inferential Analysis

Having processed responses to the survey in the form of raw frequency counts, we have applied Chi-Square nonparametric significance tests to see whether or not there are any significant relationships in between having signed a F/OSS provision contract and F/OSS adoption factors/issues such as overall F/OSS introduction experience, political influence, administrative attitude, employee training, affordability and TCO. We have observed that public officers who involve in legal F/OSS provision process face significantly more negative attitudes from their administration. The summary of that test is given in Table 1.

Table 1: F/OSS provision contracts and administrative attitude



		Administrative Attitude					
Signed FOSS Contract		Very Neg.	Neg.	Undecided	Pos.	Very Pos.	Total
	Yes	6	7	12	6	3	34
	No	3	6	4	13	9	35
	Total	9	13	16	19	12	69
	X^2 Test Statistics: Chi Square Value = 10.64 df = 4, p-value = 0.031, r^2 = 0.15						

We have combined survey items on access to expertise, support and documentation to create a single averaged 'accessibility' latent item. Figure 6 (a) shows that new combined response displays a normal like distribution with slightly positive kurtosis (*std = 0.81*) around the mean (_ = *3.33*). Then, we have grouped averaged responses to determine any opinion and experience differences in between group of respondents who have signed a F/OSS provision contract and those who have not. The normality check for each test group is performed using the normal Q-Q plots. Next, we have examined whether we can infer any significant relation in between the experience or opinion about availability of expertise, support, documentation on F/OSS and having signed a contract on any F/OSS deployment at public administrations. We have used a two-sided Welch Two Sample T-test[5]. We have observed a significant correlation (*p-value = 0.00026*). In order to examine the direction of relation we have observed response distribution of each group. It shows that those who have signed F/OSS in order to install or acquire additional F/OSS products acknowledge the easy access to expertise, support, and documentation which is necessary for organizational level F/OSS adoption.

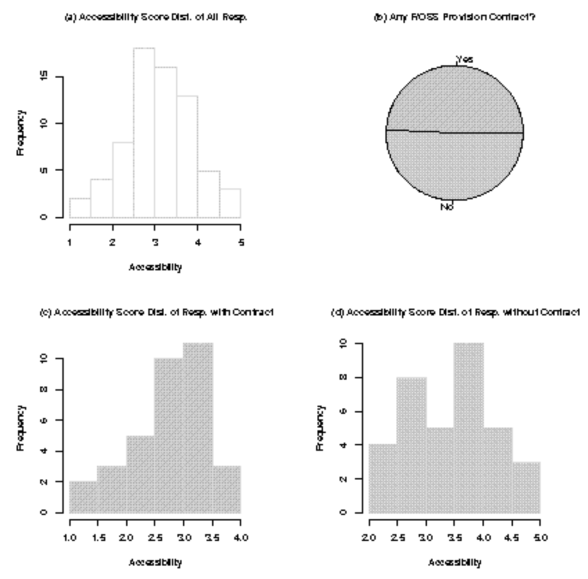


Fig. 6: Accessibility and contract relation

Following the same pattern given above we have generated another latent item 'attitude' which combines items on personal attitude and administrative attitude about F/OSS. Then, again, we have split the respondents according whether they have signed F/OSS provision contract or they have not. The t-test suggests that there is a statistically significant relationship (*p-value = 0.006)* in between developing general attitudes about F/OSS adoption and the experience of having signed F/OSS provision at public organizations. Our further analysis shows that formed attitudes are based on the particular experience of respondents. Those who have signed a contract while introducing F/OSS at their administration has apparently declare existence of more negative attitudes about F/OSS.

We finally have examined whether there is a correlation in between general F/OSS attitudes and availability of F/OSS expertise, service, and documentation. We have run nonparametric Spearman's rank correlation test[6, 7]. Our test specifically checks the availability of a positive correlation in between having easy access to F/OSS expertise, customer service, and documentation and developing a positive attitude towards F/OSS introduction. Expectedly, an estimated *p = 0.003 « _ = 0.05* indicates that there is a statistically significant positive relation but the correlation is far from perfect. This relatively low correlation *rho = 0.32* implies that administrative F/OSS attitude is not solely based on the service and it suggests that other factors such as political influence, resistance to change, and initial financial barriers should be considered.

3 Conclusions

The conclusions we derive hereby are tentative answers to those particular relationships we have investigated but it indicates that the relationships found in our survey can be expected to exist in at larger level in Europe. Yet this particular study reveals the observation that decision takers' and practitioners' perceptions about F/OSS alone do matter towards a larger F/OSS adoption at Public Organizations. However, this study urges the necessity of a larger geographical scope and higher survey responses in number with evenly distributed samplings to derive more statistically confident observations. We hope that this study will lead further detailed and specific surveys about perceptional issues at F/OSS technology adoption.

References

[1] Likert, R. (1932). "A Technique for the Measurement of Attitudes" Archives of Psychology 140, 55.

[2] Ross, K. C. (1996). Air University Sampling and Surveying Handbook. University Press of the Pacific.

[3] Ghosh, R. et al. (2002) “Free/Libre and Open Source Software: Survey and Study. Part 2B: Open Source Software in the Public Sector”

[4] Rogers, E. M. (2003). Diffusion of Innovation, Fifth Edition. New York, NY: Free Press.

[5] Welch, B. L. (1947). The generalization of "student's" problem when several different population variances are involved. Biometrika. Vol. 34, pp. 28-35.

[6] Best, D. J. and Roberts, D. E. (1975). Algorithm AS 89: The Upper Tail Probabilities of Spearman's rho. Applied Statistics, 24, 377-379.

[7] Hollander, M. and Wolfe, D. A. (1973). Nonparametric statistical inference. New York: John Wiley & Sons. Pages 185-194.

Open Source Software and Open Data Standards as a form of Technology Adoption: a Case Study

Bruno Rossi, Barbara Russo, and Giancarlo Succi
Free University of Bolzano-Bozen, Faculty of Computer Science,
Domenikanerplatz 3, 39100 Bolzano-Bozen, Italy
{bruno.rossi,barbara.russo,giancarlo.succi}@unibz.it
WWW home page: http://www.case.unibz.it

Abstract. The process of technology adoption has been studied for long time to give instruments to evaluate the best strategies to ease the introduction of technology. While the main research on Open Source Software focuses mainly on the development process, team collaboration and programmers' motivations, very few studies consider Open Source Software in this context. In this paper, we provide an overview of literature on technology adoption that can be useful to relate the concepts. We then provide a case study with historical data about file generation and usage across time to evaluate the adoption of Open Source Software and Open Data Standards in the specific case provided.

Keywords: Open Source Software; Data Standards; Technology Adoption.

1 Introduction

Open Source Software (OSS) has acquired recently a growing popularity in the software market. The free availability of source code and the freedom to modify and redistribute the source code are the main characteristics that are at the basis of its crescent popularity. Particularly in the governmental setting, these characteristics have increased the interest towards OSS. The Interoperable Delivery of European e-Government Services to public Administrations, Businesses and Citizens organisation (IDABC), identifies five aspects in this context, that can be of interest for organisations willing to adopt OSS [10]:

- political aspects, concepts related to governmental tasks, goals and responsibilities like freedom and equality, digital endurance, digital heritage and stimulation of innovation;
- economical aspects, related to cost reduction and market health;
- social aspects, in particular for education and team work support;

- managerial and/or technical aspects, in particular quality of the products in terms of stability and reliability, transparence, support and security;
- legal aspects, related to licensing and liability.

All these different point of views make the adoption of OSS inside organisations a very appealing option.

Furthermore, a concept sometimes overlooked, but frequently associated to OSS is the one of Open Data Standards (ODS). ODS are a subcategory of data standards. Data standards provide a standardised way to store different typologies of data, and emerge generally in two different ways, as output of an evolution of the market (so called *de facto* standards*)* or after being recognised by a standardisation committee (*de jure* standards). The distinction that is of our interest is between Open and Proprietary data standards. In this sense, many different definitions of ODS exist, we would like to propose the definition given by the Danish Board of Technology in 2002 [4]:

- An open standard is accessible to everyone free of charge (i.e. there is no discrimination between users, and no payment or other considerations are required as a condition of use of the standard);
- An open standard of necessity remains accessible and free of charge (i.e. owners renounce their options, if indeed such exist, to limit access to the standard at a later date, for example, by committing themselves to openness during the remainder of a possible patent's life);
- An open standard is accessible free of charge and documented in all its details (i.e. all aspects of the standard are transparent and documented, and both access to and use of the documentation is free);

As can be noted, the importance of open standards lies, in particular, in the avoidance of the commitment to a single supplier. In this paper, we review the adoption process of Open Source Software (OSS) and Open Data Standards (ODS) in an empirical case, by analysing the file generation and usage process in a single Public Administration. In this early work, we start to insert the empirical case studied in the context of technology adoption.

We first introduce two main concepts that have been discussed extensively in literature: technology adoption and lock-in. While the review we provide is not strictly related to OSS, it is necessary for the overview of the next section about technology adoption studies related to OSS. In the final part, we provide the details of the case study and the main results obtained.

2 Technology adoption

Technology adoption, diffusion and acceptance research bases its foundation on the early work of Everitt Rogers, in the book titled Diffusion of Innovations [11]. Rogers interest lies in studying the diffusion process that characterises technology adoption. In his seminal work, technology adopters are categorised according to the phase in which they make the adoption decision. The main distinction is among innovators,

early adopters, early majority, late majority and laggards. In particular, the author models the diffusion as an S-shaped curve characterised by an initial adoption speed and a later growth rate. The claim is that different technologies will lead to different adoption patterns.
Fichman & Kemerer [8] report two critical factors that influence the technology assimilation process: knowledge barriers and increasing returns.
The first effect relates to the effort necessary to acquire the necessary knowledge and skills to properly adopt a certain technology. This effect leads to what are known as knowledge barriers [2,8].
The second phenomenon, reports that the adoption of certain technologies is subject not only to supply-side benefits due to economies of scale [12], but also to a demand-side effect called increasing returns effect [1]. The effect leads to an increase of the utility in adoption for each successive adopter, based on the number of previous adopters. Arthur goes further in this analysis, claiming that economy can become locked-in to a technological path that is not necessarily efficient, not possible to predict from usual knowledge of supply and demand functions, and not easy to change by standard tax or subsidy policies [1]. In this sense, it may not be possible to easily switch from a certain technology once a certain critical level of adoption has been reached.

Considering OSS, there are not many studies that evaluate OSS from this point of view. An interesting overview is given in [5], where following the "context for change methodology" defined in [6], factors that lead adoption process are categorised in technological, organisational and environmental.

Glynn et al. [9] developed a framework for assimilation based on four categories: external environment, organisational context, technological context and individual factors. The framework is then applied to a large-scale survey.
Bitzer and Schroder [3], analyse the innovation performance of Proprietary and Open Source Software, showing the results of the competition between the two software typologies in different market settings. The focus is more on innovation that on the adoption process itself.

Economides [7] studies the incentives that lead to platform innovation. A case study between Linux and Windows is provided.

3 A case study of OSS migration

To provide some real data about a concrete case of OSS and ODS adoption process, we consider an experimentation that took place during an experimental migration from Microsoft Office to OpenOffice.org in one medium-size European Public Administration. The users involved were 100. Data have been collected by means of the PRO Metrics (PROM) software [13], software that permits to collect metrics about software usage, and FLEA (FiLe Extension Analyzer), software that allows collecting information about the data standards available on the target system.

Operations performed were the installation of OpenOffice.org in parallel with the available version of Microsoft Office, installation of the PROM agent, scan of the file-system with FLEA, training of users and support.

4 Main results

We report the main result from the analysis of data standards and software usage. In table 1, we show the total number of all the data standards collected at the beginning of the experimentation, divided per category.

Table 1. Total number of data standards detected by the data collection software

Text Documents		Graphic Format		Database	
DOC	310648	BMP	6908	DB	3361
DVI	0	GIF	36259	DBF	6865
PDF	12518	JPEG	83143	MDA	10
PS	656	PNG	2173	MDB	2170
RTF	6185	SVG	0	Music	
SXW	160	TGA	0	MP3	967
TEX	1	TIFF	11061	RA/RM/RAM	4
TXT	10422	Drawing		Movie	
Spreadsheets		DWG	36051	AVI	265
SXC	47	DXF	411	MOV	227
XLS	60267	SXD	9	MPEG	77
Presentations		Web		SWF	232
PPT	2541	CSS	1370		
SXI	27	HTML	16057		
Compression		XHTML	0		
ACE	1	Data Exchange			
ARJ	37	CSV	64		
GZ	19	DTD	6		
RAR	43	SDXF	0		
TAR	0	XML	483		
ZIP	5338				

As we can see, some data standards are largely predominant in their category, like DOC (Microsoft Word) documents, or XLS documents (Microsoft Excel). The former accounts for 91,21% of the files in the category, while the latter 99,92%. Also the ZIP format is largely dominant, with a percentage of 98,16%. If we use the Shapiro & Varian [12] categorisation of switching costs and consider the information and databases category, we can evaluate that a complete migration and adoption of the platform can be costly, due to the effort necessary required by the conversion of a large amount of documents.

We further studied the evolution of file generation across time, in figure 1 we show the generation of DOC and XLS documents, from the data collected the more representative for proprietary formats.

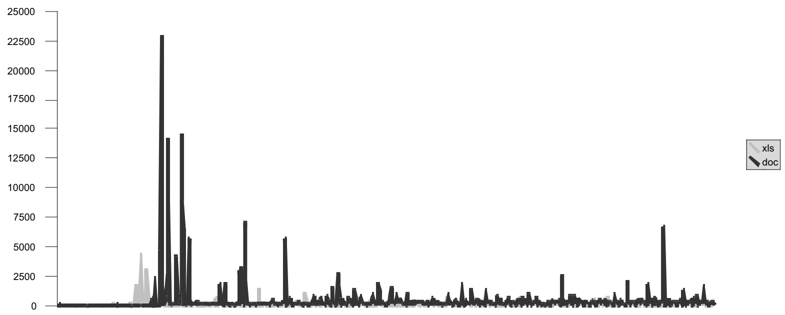


Fig. 1. Evolution of DOC and XLS files created by users during years, on x-axis time in days, on y-axis number of files generated

While the figures are given for representational purposes, we may note an interesting phenomenon that emerges by analysing data of many data standards. The creation process of documents is very consistent once a critical level of adoption has been reached. Once a large set of base documents has been constituted, the creation process somehow fades, as the activity of users will be constituted in small part also by the modification of already available documents.

5 Conclusions

While OSS research community is concerned mostly with studying the team and collaboration dynamics of the development process, OSS and ODS have still to be well studied as a form technological innovation. We overviewed some of the technology adoption literature that may be useful in this sense and some recent works that manage to insert OSS in this context. We further considered ODS as an important and often overlooked instrument that has to be associated to OSS when considering its adoption. We studied as a case study, the evolution of a migration to OSS in the office automation field, considering data standards as a sign of the presence of possible lock-in phenomena. The data analysed show the commitment of the organisation under study to proprietary data formats, in particular in the office automation category.

Acknowledgements

This work has been partially supported by COSPA (Consortium for Open Source Software in the Public Administration), EU IST FP6 project nr. 2002-2164.

References

[1] Arthur, W.B. (1989). Competing Technologies, Increasing returns, and lock-in by historical events. Economic Journal, 99, 116-131.

[2] Attewell, P. (1992). Technology Diffusion and Organizational Learning: The Case of Business Computing. Organization Science. 3(1), 1-19.

[3] Bitzer, J. and P. J. H. Schroder, (2003). Competition and Innovation in a Technology Setting Software Duopoly. DIW Discussion Paper No. 363.

[4] Danish Board of Technology. (2002). Definition of open standards. Retrieved, 14th January 2007, from http://www.oio.dk/files/040622_Definition_of_open_standards.pdf

[5] Dedrick, J., West, J., (2004). An Exploratory Study into Open Source Platform Adoption, HICSS, p. 80265b, Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 8, 2004.

[6] Depietro, Rocco, Edith Wiarda and Mitschell Fleischer, (1990). The Context for Change: Organization, Technology and Environment, in Tornatzky, Louis G. and Mitchell Fleischer, The processes of technological innovation. Lexington, Mass.: Lexington Books, 1990, pp. 151-175.

[7] Economides, N., Katsamakas, E. (2006). Linux vs. Windows: A comparison of application and platform innovation incentives for open source and proprietary software platforms, in Juergen Bitzer and Philipp J.H. Schroeder (eds.) The Economics of Open Source Software Development, Elsevier Publishers, 2006

[8] Fichman, R. G., & Kemerer, C. F. (1997). The Assimilation of Software Process Innovations: An Organizational Learning Perspective. Management Science. 43(10), 1345-1363.

[9] Glynn, G, Fitzgerald, B and Exton, C. (2005) Commercial adoption of open source software: an empirical study, Proceedings of International Conference on Empirical Software Engineering, Noosa Heads, Australia, Nov 2005.

[10] IDABC, The Many Aspects of Open Source. Retrieved, 14th January 2007, from http://ec.europa.eu/idabc/en/document/1744

[11] Rogers, E. (1995). Diffusion of Innovations. N.Y.: The Free Press.

[12] Shapiro, C., & Varian H.R. (1999). Information Rules: A Strategic Guide to the Network Economy. Harvard Business School Press.

[13] Sillitti A., Janes A., Succi G., Vernazza T. (2003). Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data, in proceedings of EUROMICRO 2003, Belek-Antalya, Turkey, 1 – 6 September 2003

Free/Open Source Software Adoption, Public Policies and Development Indicators: An International Comparison

Harald Schmidbauer[1], Mehmet Gençer[2], and Vehbi Sinan Tunalıo_lu[2]
1 Istanbul Bilgi Üniversity, Department of International Finance, harald@bilgi.edu.tr
2 Istanbul Bilgi Üniversity, Department of Computer Science, {mgencer,vst}@cs.bilgi.edu.tr

Abstract. Despite the growing body of research on the inner workings of FOSS development, there are few studies on its relation with broader developments in society. In this study we have attempted a preliminary investigation of (1) how FOSS prevalence is related to economic and human development indicators of countries, (2) whether public policies regarding FOSS emerge in a consistent relation with these indicators in several clusters of countries constructed from the United Nation's human development index, and (3) the relation of software piracy to development indicators. Our results point to relative significance of non-economic factors in FOSS adoption, lack of consistent policies among public agencies, and irrelevance of non-economic factors on software piracy. In addition, the study demonstrates the possibility of developing FOSS indices for larger scale diagnosis and strategizing.

1 Introduction

The development of policies towards Free and Open Source Software (FOSS), whether public or private, requires a better understanding of how FOSS adoption interacts with various aspects of society. While the growing body of research on FOSS focuses on community, governance, and coordination of production activity [6,4], or relations to wider software industry [3,11], studies on relation of FOSS with broader developments in society and factors which affect its adoption are seldom [1,2].

In this study we have attempted a preliminary investigation of these relations. We first address the following empirical questions: (1) how FOSS adoption is related to economic and human development indicators of countries, and (2) whether public policies regarding FOSS emerge in a consistent relation with these indicators in several clusters of countries. In addition, (3) relation of software piracy to development indicators is considered in order to highlight deficiencies in policies involving proprietary software. Building on our results and research elsewhere, we then discuss policy choices accounting for economic reality.

Corresponding methods for our investigation are as follows:

1. For an international comparison of FOSS adoption, we particularly tried to assess the variability in the occurrence of FOSS across countries on the basis of several

variables related to human development, economics and transparency, and the Internet infrastructure. Adoption of FOSS in a country is operationalized as the number of hits in the country-specific Google search of the keyword "open source". This variable will obviously be correlated with the size of a country in terms of its population.

2. To assess the relation of public policies towards using FOSS with development indicators, we have attempted to relate it to a classification of countries we have developed on the basis of cluster analysis. FOSS usage in public agencies is operationalized through the server software type used in several public bodies in countries.
3. Deficiencies in policies involving proprietary software is examined through an analysis of piracy data for countries in relation to development indicators. For this purpose piracy rate data published by BSA was checked for its correlation to human development, economics and transparency indicators.

Our results point to (1) relative significance of non-economic factors in FOSS adoption, (2) lack of consistent policies among public agencies towards FOSS, and (3) strong dominance of economic factors and irrelevance of non-economic factors on software piracy. The finding that FOSS adoption is correlated to human development index, which is defined as "a process of enlarging people's choices" [10], is supportive of the claims that public promotion of FOSS through information campaigns can indeed be effective [1]. Furthermore, we argue that given the economic conditions in relatively less developed countries, public intervention in the software market through promoting open source is not only favorable [2], but possibly inevitable if sensible conditions in software market (such as reduced piracy) are desired. On the other hand lack of consistent public policies indicates lack of awareness among policy makers towards the potential of open source software, in effect disengaging large communities from enjoying its benefits.

The following three sections lay out quantitative data, models and results for relating development indices to FOSS adoption, public policies towards FOSS and software piracy, respectively. A discussion of results in section 5 attempts to interpret these results, followed by conclusions.

2 FOSS Adoption

Our investigation of the FOSS adoption is based on regression models with Google counts of "open source" as dependent variable and a selection of independent variables from the following list: population, gross domestic product (GDP), human development, index (hdi), transparency index (tri), Internet usage, relative Internet usage, number of Internet hosts in the country.

In order to account for the impact of economic reality, we use GDP per capita, the transparency index and Internet usage as proxies to the adoption of FOSS. We also relate FOSS adoption to human development. We choose the human development index (despite ongoing debates, see, for example [5,8]), which is published by the United Nations Development Program (UNDP) [10].

A full regression model, with all variables included as regressors, and fitted to 116 cases (countries) for which the entire data set was available, has an explanatory power (i.e., R^2) of 67%, but leaves many variables insignificant. Using stepwise procedures to fit a reduced model with fewer, and significant variables, leads to the following model, fitted to 129 cases (The numbers in parenthesis are the t values of the estimates):

$$\begin{aligned} \texttt{log(log(count.open.source))} &= \underset{(-3.13)}{-1.546} + \underset{(3.37)}{0.8155} \times \texttt{log(log(population))} \\ &\quad + \underset{(4.11)}{0.0485} \times \texttt{log(internet.hosts)} \\ &\quad + \underset{(5.44)}{1.3070} \times \texttt{hdi} \end{aligned}$$

This model also has an R^2 of about 67%, while retaining only significant regressors. A comparison of the regressors in this model with the list of variables in the table above reveals that Internet usage, relative Internet usage, the transparency index, and GDP per capita are not significant in explaining the variability of Google counts of "open source" across countries.

It may come as a surprise that GDP per capita has so little explanatory power on the adoption of FOSS compared to `hdi`. To reassess this result, another regression model, with GDP per capita substituted for hdi, can be fitted to the data set. The explanatory power of such a model is substantially smaller than the model cited above; its R^2 is reduced to 61%. We can thus conclude that economic well-being can indeed explain a certain share of variability contained in the dependent variable, but the explanatory power is further enhanced by considering non-economic factors. This highlights the importance of non-economic factors in accounting for the prevalence of FOSS.

3 Public Policies

Is there a pattern in the server software variety used by governmental organizations across countries? To provide an answer to this question, we have looked into the server software choice (open source vs proprietary) in several prominent public bodies of countries; these are the bureau of statistics, the central bank, the finance regulator, the foreign ministry, and the postal services. In addition we have clustered countries with respect to the variables discussed earlier. Finally, we have tested the null hypothesis that cluster membership and the server software variety are independent.

A cluster number of 5 was found optimal to account for differences between the countries. The five clusters (based on a k-means cluster analysis with standardized variables) can be summarized as follows:

- Cluster 1: Low relative Internet usage, mid-range hdi; mid-range open-source count reflecting the mid-range population of the country.
- Cluster 2: Low relative Internet usage, mid-range hdi; low open-source count reflecting the smaller size of the country as compared to Cluster 1.
- Cluster 3: Low relative Internet usage, mid-range hdi; higher open-source count reflecting the larger size of the country as compared to Cluster 1.
- Cluster 4: Highest relative Internet usage, highest hdi; very high open-source count, almost irrespective of the population size.
- Cluster 5: Very low relative Internet usage, very low hdi.

Figure 1 shows a classification tree. It illustrates how the cluster membership of a country is determined: if the condition on a fork is fulfilled, the left branch applies, and vice versa.

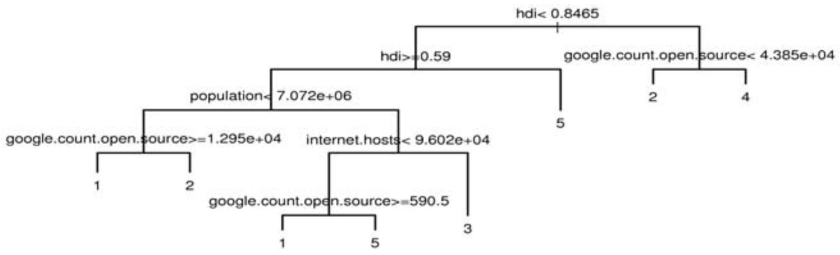

hdi< 0.8465
hdi>=0.59
google.count.open.source< 4.385e+04
2
4
5
population< 7.072e+06
google.count.open.source>=1.295e+04
internet.hosts< 9.602e+04
1
2
google.count.open.source>=590.5
3
1
5

Fig. 1: Classification tree for clusters of countries

Table 1 shows the server software counts for public bodies in different clusters, with countries for which no data were available omitted. Since the main difference between Clusters 1, 2 and 3 is population size, it is justified to collapse them into a single cluster and increase the power of the _^{2} test. The p-value of the null hypothesis that "server software choice and cluster membership is independent", as shown in the last row of the mentioned table, shows that it is rejected only in the case of the bureau of statistics.

Table 1: Cluster membership and server software variety (f = free, p = proprietary)

	central bank		bureau of statistics		finance regulator		foreign ministry		postal services	
cluster				p	f	p	f		[illegible]	[illegible]
1, 2, 3	1	9		17	12	16	11		[illegible]	1
4		2		18	13	11	16		4	0
5				1	3	2	3		[illegible]	[illegible]
p-value	0.72		0.04		0.63		0.52		0.64	

A related question is: Is there evidence that the variety of server software used in one organization of a country is associated with that used in another one? We found no case in which the hypothesis of independence of server software variety in different organizations could be rejected. In this sense, we found no evidence of a consistent server software policy pattern within countries.

4 Software Piracy

In what way is software piracy connected with the variables listed above, and with FOSS prevalence in particular? Insight into these matters may have consequences concerning policies to reduce software piracy supported by a policy to promote FOSS. The most comprehensive data for software piracy across countries comes from Business Software Alliance (BSA), despite debates regarding its reliability. Fitting a regression model with privacy as dependent variable and independent variables logged GDP per capita and transparency index to the 66 available cases leads to

$$\texttt{piracy} = \underset{(8.03)}{201.37} - \underset{(-4.30)}{13.263} \times \texttt{log(gdp.pc)} - \underset{(-4.04)}{3.6566} \times \texttt{tri}$$

This model has an R^2 of 0.787. The fact that the human development index shows no significance reveals dominance of economic factors on software piracy.

Another far-reaching aspect with regard to policy finding may be the correlation between piracy and FOSS adoption (operationalized as Google counts of “open source”) across countries. This correlation is a moderately distinct _0.44. This correlation may be spurious, however, since both variables are closely linked to GDP. Indeed, the partial correlation between piracy and FOSS adoption, controlling for GDP per capita, turns out to be close to 0. A preliminary conclusion is that promoting FOSS adoption, without accompanying measures, need not be a cure against software piracy.

5 Discussion

As explained in the previous section, we found no evidence that governments pursue a server policy. Based on macro-economic models, Comino and Manenti have recently pointed out that regarding open source software “there may be a substantial mass of uniformed consumers, leading to market failures that may justify government intervention” [1]. Results of our empirical comparison also directs in similar ways that non-economic factors are an important determinant in harvesting of potential advantages FOSS provides. But more importantly it highlights that development of “free and open source indices” of better precision can be very empowering for all parties in the game as they as their interpretation can help in policy making.

6 Conclusions

In an effort to assess the determinants of the prevalence of Free and Open Source Software (FOSS), we have tried to account for the variability in Google counts of the keyword "open source" across countries worldwide using a regression model with several regressors characterizing human development, the economic situation, and Internet infrastructure of the countries considered. It turned out that the human development index, population, and the number of Internet hosts are significant, while Internet usage is not. A cluster analysis of the country data pointed at the interplay of human development and FOSS prevalence in public agencies. We found that the bureau of statistics is the only example among several prominent public organizations of a country whose server software variety (free or proprietary) is significantly related to the cluster membership of countries.

We conclude from our models that GDP per capita is less important in accounting for prevalence of FOSS. It is rather information campaigns and educational incentives that may be conducive to widespread use of FOSS.We mentioned earlier that FOSS introduced its own mode of production, in which human capital plays the most important role. The study by Streeten [12] emphasizes that "human beings are both ends in themselves and means of production". With respect to that, we suggest that FOSS be further discussed under the aspect of its contribution to human development.

References

1 Comino S, Manenti FM (2005) Government Policies Supporting Open Source Software for the Mass Market. Review of Industrial Organization 26(2):217-240.

2 Dalle JM, Jullien N (2003) Libre software: turning fads into institutions? Research Policy 32(1):1-11.

3 Garud R, Kumaraswamy A. (1993) Changing competitive dynamics in network industries: an exploration of Sun Microsystems' open systems strategy. Strategic Management Journal 14(5):351-369.

4 Hippel Ev, Krogh Gv (2003) Open source software and the private-collective innovation model: Issues for organization science. Organization Science 14(2):209-223.

5 Kelley AC (1991) The Human Development Index: ``Handle with Care''. Population and Development Review 17(2):315-324.

6 Kogut B, Metiu A (2001) Open source software development and distributed innovation. Oxford Review of Economic Policy 17(2):248-264.

7 Srinivasan TN (1994) Human Development: A New Paradigm or Reinvention of the Wheel? The American Economic Review 84(2):238-243.

8 Streeten P (1994) Human Development: Means and Ends. The American Economic Review 84(2):232-237.

9 Transparency International (2004) Annual Report 2004 - The Coalition Against Corruption. Transparency International

10 UNDP (1990) Human Development Report 1990. Oxford University Press, New York, Oxford.

11 West J (2003) How open is open enough: melding proprietary and open source platform strategies. Research Policy 32(7):1259-1285.

Levels of Formality in FOSS Communities

Andrew Schofield[1], and Professor Grahame S. Cooper[2]
1 Salford Business School, University of Salford, Salford, M5 4WT, UK, a.j.schofield@pgt.salford.ac.uk,
WWW Home page: http://www.postgrad.isipartnership.net/~aschofield/
2 School of Computing, Science, and Engineering, University of Salford, Salford, M5 4WT, UK, g.s.cooper@salford.ac.uk,
WWW Home page:
http://www.cse.salford.ac.uk/profiles/profile.php?profile=G.S.Cooper

Abstract. One of the aspects of Free and Open Source Software (FOSS) which may act as a significant deterrent to its adoption, is the method used to collaboratively develop the software and provide support through the use of communities. It is not until this method is examined more closely that its many advantages can be realised. The method can, however, seem very disorganised especially when compared with traditional proprietary development styles. A key difference between these two development approaches lies in the management of projects, and perhaps as a consequence, in the level of formality in the community environment. This paper presents the results of empirical survey research investigating FOSS community participants' views on the level of formality in FOSS, and the way in which this affects both development and support provision activities. The paper then concludes by analysing what can be learnt from the participant's views.

1 Introduction

Despite the many success stories and research studies demonstrating the advantages and capabilities of FOSS, the stereotypical view of ideas and code being thrown around within disorganised communities, still has the potential to deter individuals and organisations from using or developing it. In actual fact, these communities are arguably the most important element of FOSS. Much research has been done on FOSS communities [1-4, 6-8, 10] demonstrating that there is a general framework that communities follow [8, 9]. However, differences in working methods and styles of approaches become very apparent when comparing FOSS communities. These are also seen as major differences between FOSS and proprietary closed-source development [5, 7].

This paper contributes to the knowledge in this area by presenting the results of empirical survey based research, which collected FOSS community participants' experiences and views on the level of formality within FOSS communities, and its

Please use the following format when citing this chapter:

Schofield, A. and Cooper, Professor G.S., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 337–342.

subsequent effect. As these communities are often quite complex, the research was split into two sections, one focusing on the support aspects of the communities, and the other focusing on development activities.

2 Research Method

As its primary data collection tool, the research used an on-line survey consisting of open questions designed to collect qualitative data. The sample set for the research consisted of Linux, BSD, and Open Source user and/or interest groups, hereafter referred to as Linux User Groups or LUGs. The research targeted LUGs within the UK, US, Italy, Germany and Canada, as these were the countries with the highest number of LUGs. In total 392 responses were received from the various countries, 48% of which were from developers. It must be pointed out that LUGs cannot be considered as absolutely representative of FOSS communities. However, the survey was particularly designed to collect members' perspectives of FOSS communities in general, and LUG members are also likely to participate in other communities as well. Additionally it should be noted that the sample consists of FOSS community members, and although some may have proprietary software experience, it should be recognized that a bias towards FOSS software will exist. The paper's purpose, however, is not to specifically compare the formality of FOSS and proprietary approaches, but rather to examine the effect of formality on FOSS communities.

3 Research Findings

3.1 FOSS Community Formality

The first section of the survey dealt with the general concept of formality in FOSS communities. Participants were asked their opinions about the level of formality in the working practices of FOSS communities, when compared to the proprietary software approach. The responses, grouped by topic, are summarised below:

- **Project/Community Dependant**: A significant number of respondents wrote that the formality of a project or a community is very specific, and that a generalised statement that describes all of them is not possible.
- **A Mixture of Elements**: Respondents pointed out that communities can be viewed as informal in terms of them being open for anyone to participate, however, in terms of the management of the final product, FOSS communities could be seen as very formal. Others pointed out that how formal a community is depends on how it is led, and that the formality of the methods used in a community project are needed only for project management purposes

- **Informal Interaction**: Many respondents wrote that the interactivity side of FOSS communities is fairly informal (i.e. discussion forums, etc). Others added that formality depends on how well one knows the project leader, and presumably the other community members.
- **Depends on the Projects' Complexity**: Many respondents stated that the formality of a community depends on the project's size, scope, complexity, and maturity. Consequently, a large project requires a high degree of formality to keeps things under control. However, for each of these large projects, there are many smaller sub-projects that are far less formal.
- **Theory Versus Practice**: Many respondents wrote that in both the case of FOSS and proprietary software, formal rules and guidelines may be set down, but are seldom followed. Additionally, community based and company based software development often operate with the same rules, but it is how these rules are enforced that differentiates the two types.
- **The Freedom of FOSS**: Some respondents stated that FOSS is more informal because of its underlying ethos. They stated that those involved in FOSS do not want to be restricted to a formal and controlled system.

3.2 Effects of Formality on FOSS Support Activities

Survey participants were then asked to comment on how the level of formality in FOSS communities affects support provision.

- **A Deterrent to the use of FOSS**: Respondents posted that the unwritten rules and etiquette used in FOSS forums could make people feel unwelcome. Many also felt that frequent arrogance among knowledgeable community participants leads to ‘newbies’ being unwelcome.
- **Project Dependant**: Some wrote that the effect would depend on the size of the project. Large projects having good support because of the number of people involved, and smaller projects having poor support as the fewer members will have less time free to provide support.
- **Formality Improves Support**: Several respondents observed that communities with strict and formal working practices have very good support, particularly documentation. Likewise, those less formal communities were often found to be lacking in support.
- **Ease of Using Forums**: Other respondents felt that the informal nature of FOSS community forums facilitates interaction. Contrary to the comments left above, it was suggested that the equality and lack of a hierarchy, would make members willing to help one another. The importance of interplay between experts and ‘newbies’ was also emphasised as a positive factor.
- **Direct Contact with Programmer(s)**: Several participants wrote that FOSS communities allow direct communication to the actual writer of the code. This clearly has advantages as no-one could provide better support, and the questions asked might also stimulate ideas on further code development.

- **Enjoyment of Support Forums**: Comments were left stating that the informal nature of support forums is enjoyable, but that anyone requesting support in an inappropriate manner is likely to receive an unhelpful reply.

3.3 Effects of Formality on FOSS Development Activities

Finally, respondents were asked to comment on how the level of formality affects software development within FOSS communities.

- **The Big Picture**: Without some control or steering group, respondents believed that programmers would do what they personally think is best, which may not be what is best for the overall project.
- **Very Informal Management**: Several respondents stated that projects with an extremely relaxed approach lead to either unusable products or a forking of the projects, leading to several very similar products.
- **Natural Formality**: Some respondents wrote of projects adapting and finding their own level of formality. One respondent referred to this as a community's *"natural formality"*.
- **Development Feedback**: Several survey respondents pointed out that the detection of bugs, and even design ideas can originate from questions asked on a support forum, and how this is supported by an informal environment.
- **Higher Formality for Larger Projects**: Many respondents stated that they felt larger projects required a more formal structure to manage all the code submissions, while others felt that this was not a problem with the use of versioning software.
- **Openness in Development**: It was pointed out that the open approach can lead to arguments and disagreements, especially in very informal and undisciplined projects.
- **Informal Development Rules**: Although most respondents felt that at least some formal structure was required, some felt that the lack of rules was a good thing, allowing FOSS development to release leading edge software quicker. This was also facilitated by the unrestricted communication and the lack of red tape.
- **Informality Breeds Innovation**: Many respondents thought that the freedom of an informal environment helps developers to be dynamic, innovative, and have the freedom to experiment, and that a formal environment stifles creativity.
- **More control needed**: Although it was a minority view, some felt more control or planning was needed. They felt there was too much discussion about software functionality and project direction, and that FOSS communities often have problems related to clashing egos and methods.
- **Deterrents to Involvements**: A disadvantage pointed out was that informal environments could deter those who have previously worked in a more structured environment. Others however, felt that it would encourage participation

- **Communication Leads to Results**: Several wrote that FOSS communities facilitate communication and that the informal approach makes it easier for developers to work together.
- **Formal Practices at the Right Time**: Some respondents felt that FOSS development should adopt the most suitable level of formality for the phase of the development. As a project grows, and more become involved, a *"benevolent dictator"* is needed. Some referred to this as a Darwinist approach, because only the *'fittest'* submissions are accepted.
- **Disadvantages of Voluntary Work**: Some survey respondents felt that more effort is put into developing software that is fun to write, rather than the more mundane or boring applications. To quote one respondent *"that's why we only have 3 office suites but about 42,000 music players"*.
- **Informality is Good for Growth**: A few respondents wrote that although informal approaches are good for recruiting new members, they are often then less motivated to work than in a formal environment.

4 Conclusions

The results allow us to further define the concept of formality with regards to FOSS. We can first separate formality into more specific important factors, which we shall call 'managerial formality' and 'cultural formality'. Managerial formality refers to aspects of formality which are for the purposes of organisation and structure. These manifest themselves as rules and regulations concerned with the support and particularly the development of FOSS. Many respondents wrote of the importance of formal management, particularly for large projects with many people involved and during phases of the development cycle where decisions about the direction of the development are made. Cultural formality refers to the level of formality which exists between the participants of the community. This is evident from the discussion boards and mailing lists of a community and is defined by the members themselves, with a possible influence from the managerial formality. The essential difference is that the former is imposed, or perhaps suggested, by the community leaders, while the latter comes into being or develops from the personalities and actions of the participants.

From the analysis of the responses, it seems that the predominant view is that managerial formality improves both support and development activities. Nevertheless, many survey participants warned of the dangers of an environment which was too formal. Likewise, cultural formality promotes freedom and innovation, but can be off-putting to 'newbies' or those used to formal practices.

5 References

1 Ghosh, R., A., Glott, R., Krieger, B., Robles, G. (2002), "Survey of Developers", Free/Libre and Open Source Software: Study and Survey, International Institute of Infonomics, University of Maastricht, The Netherlands, Available at: http://www.infonomics.nl/FLOSS/report/ , (Accessed May 2006)

2 Hann, I. H., (2004) "Why Developers Participate in Open Source Software Projects: And Empirical Study", Twenty-Fifth International Conference on Information Systems.

3 Hertel, G., Niedner, S., Herrmann, S. (2003), "Motivation of Software Developers in Open Source Projects: An Internet-based Survey of Contributors to the Linux Kernel", Research Policy, Special Issue on Open Source Software Development, Available at: http://opensource.mit.edu/papers/hertel.pdf ,(Accessed February 2004)

4 Lakhani, K. R., Wolf, R.G. (2003), "Why Hackers Do What They Do: Understanding Motivation Effort in Free Open Source Software Projects", MIT Sloan School of Management Working paper, Available at: http://freesoftware.mit.edu/papers/lakhaniwolf.pdf, Accessed (February 2004)

5 Moody, G. (2001), "Rebel Code: How Linus Torvalds, Linux and the Open Source Movement Are Outmastering Microsoft", The Penguin Press, England

6 Oh, W., Jeon, S., (2004) "Membership Dynamics and Network Stability in the Open-Source Community: The Ising Perspective" Twenty-Fifth International Conference on Information Systems.

7 Pavlicek, R. C. (2000), "Embracing Insanity: Open Source Software Development", Sams Publishing, USA

8 Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., Lakhani, K., (2005) "Understanding Free/Open Source Software Development Processes", Available at: (http://www.ics.uci.edu/~wscacchi/Papers/New/SPIP-FOSS-Intro-Dec2005.pdf) (Accessed December 2005)

9 Schofield, A., Mitra, A. (2005), "Free and Open Source Software Communities as a Support Mechanism", UK Academy of Information Systems conference 2005, Newcastle, UK

10 Zhang, W. & Storck, J, (2001) "Peripheral Members in Online Communities", Americas Conference on Information Systems, Boston, USA

Stakeholder value, usage, needs and obligations from differnet types of F/LOSS licenses

Darren Skidmore
Monash University, Melbourne Australia.
darren.skidmore@infotech.monash.edu.au

Abstract. This paper discusses different types of Stakeholders of F/LOSS, their needs, the value, usage, and obligations that stakeholders have for different types of F/LOSS licenses. Stakeholders include **Developers**: Individuals, Projects, Embedded Systems. **Vendors**: Dominant, Niche, F/LOSS. **Packagers. End Users**. **Organisations:** Disseminators, Internally, Externally Used .

Key words: F/LOSS, Licenses, Stakeholder analysis, Obligations, Value, Usage

1 Introduction

This paper, briefly, explores the perspectives of different stakeholders their requirements / constraints and usage of types of Free / Libre and Open Source Software (F/LOSS) licenses. The increase in the number, the reach and range of F/LOSS applications, as well as the types of stakeholders creates a discussion about the types of licenses which suit different stakeholders, and their needs. This is not a paper about specific licenses, but about broad types of licenses and the generalized aspects of the types, individual licenses still need to be investigated for their specific terms.

2 F/LOSS licenses

The license of the software determines what rights and obligations govern the usage of the source code. The most common F/LOSS licenses in use are the GNU GPL, and GNU LGPL and the BSD licenses[1]. These match to three main general types of licenses: reciprocal, linking and non-reciprocal licenses. Reciprocal license requires

[1] Using data from FLOSSMole project - http://ossmole.sourceforge.net/ . Because of space limitation this cannot be shown, the FLOSSMole has SQL queries that generate the data.

Please use the following format when citing this chapter:

Skidmore, D., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 343–348.

software that uses source code licensed under a reciprocal license must be licensed under the same license [1]. A linking license allows for an application to link to a library or application for functionality without requiring the linkee application to be licensed under the license of the linkor application. A non-reciprocal license does NOT require an application to be relicensed under the license of the original code. Other license types exist such as obligation licenses which require specific conditions or impose limits. Table 1 gives a brief description of a taxonomy of license types drawn from [2].

Table *1.* Different types of F/LOSS licenses from [2]

License Type	Brief Explanation	Example
Traditional	Considered to be the more traditional types of F/LOSS licenses	
Reciprocal	Require derivatives to follow original license	GNU GPL
Non-Reciprocal	NOT required to follow original license	BSD
Linking	Allow other code / applications to link	GNU LGPL
Dual	Different conditions for different types of Use	MySQL
Quasi Open Source	These may or may not have other considerations attached to them	
Obligation	Have an obligation or restriction	Squiz.Net
Morality	Moral conditions restricting license use	HESSLA
Viewable Source	Allows viewing of source code, but not usage	Ms-RL
Membership	Usage of source within a select community.	Avalanche Corp
Support	Not software licenses but assist and support other aspects of F/LOSS	
Content	For documentation and support information.	Creative Commons
Open Standards	For Standards for interoperability	
Public Domain	Anyone can take and use the work.	
Closed Source	No access to source code.	Microsoft EULA

3 Stakeholders

The stakeholders below are explained, in each section. The **incoming** licenses are those that govern the source code and / or applications that the stakeholders use. The **outgoing** licenses are the licenses that stakeholders use for governing the source code / applications they produce for others to use. It is possible that the licenses of the incoming and outgoing source code are the same, different or available under multiple types of licenses.

Table *2.* List of Stakeholders in F/LOSS

Developers	
	Individual (developers)
	Project (developers)
	Embedded Systems (developers)
Vendors	
	Dominant (vendors)
	Niche (vendors)
	F/LOSS (vendors)
Packagers	
End Users	
Organisations	
	Disseminators (organisations)
	Internally Used without Development (organisations)
	Internally Used with Development (organisations)
	External Shared with Development (organisations)

Due to page limitations, the discussion is brief, which is a limitation of the paper. Many of the stakeholders will share issues, but raised here are those important to that stakeholder. Individually a business decision may be made about the license of the source code against alternatives or a specific philosophical or ideological choice may decide the license choice.

3.1 Developers

Developers are historically the users and participants in F/LOSS. In this paper, developers are generally individuals or collections of individuals.

For incoming code developers may want to develop the code on their own, but may procure code to make their own work easier, solve problems, learn, or to provide functionality. Developers might prefer using a non reciprocal style license [3] or Linking licenses.

For outgoing license, the developer might allow anyone to do what they wish, with a non-reciprocal license, or may ensure others share their modifications, or do not profit from their work and use a reciprocal license. Depending upon the incoming licenses used there may be no choice.

Individual (developers)

Individual developers want to use software for their own use or for small number of people. Since they are smaller, they may have to accept impositions of incoming licenses, due to lack of resources.

Project (developers)

The Projects are where developers and others have organised themselves into a project, perhaps large (e.g. KDE, Apache), or small project. The issues for Project developers especially for incoming licenses are similar to that of the Packagers. Some problems are in the engineering of the project, which code is being added, and from whom, or where did the code come. Where all of the code is being developed from scratch then the authors of the code, are able to license that code as they wish. Otherwise where there is a combination there are issues of management of license mixing.

For outgoing licenses the issues are similar to that of the generic developers. The project might create their own license, to protect or enable aims of the project, eg protect trademarks [4], morality / ethical considerations, [5] or commercial control [6].

Embedded Systems (developers)

Embedded systems are generally specially built to carry out a task, such as mobile phones, lift controllers. With outgoing licenses, they may use a non-reciprocal license to increase the adoption, to increase adoption by purchasers of the embedded system; another option might be dual licenses.

3.2 Vendors

A vendor has a commercial focus with a product or range of products. Vendors might be dependent on other applications in the software stack, needing to build a supporting application or rely on third parties products.

Dominant (vendors)

Dominant vendors have control of large sections of a market. F/LOSS can be used in a mixture of tactical and strategic ways (i.e. participation in the community; sharing development costs, bug fixing and innovation). Vendors may participate in F/LOSS activities of software products, to lower their user's costs, while not interrupting their own revenue streams.

The general usage by Dominant vendors seems to be of reciprocal licenses, since this keeps changes and innovation open. There might also be use of some obligation licenses, but this is generally seen in Niche (vendors).

Niche (vendors)

Niche vendors are generally small to medium vendors, who may do work on request, and / or may have a niche product in the market.

They may need incoming functionality to fit into their software stack and so linking licenses might be used. Where they cannot link but must incorporate Non-Reciprocal licenses are probably the preferred choice.

With outgoing licenses, the use of linking or non-reciprocal licenses, might encourage others to use their applications, with the Niche vendor being able to pickup maintenance work, or advertise their expertise. They might also use an obligation license, to assist their organisational aims.

F/LOSS (vendors)

F/LOSS vendors are those vendors who primarily use F/LOSS, as part of their offerings to the market, in general they would be similar to Niche (vendors), but should be more sophisticated in their use of F/LOSS.

3.3 Packagers

Packagers are people or projects which package up FLOSS applications into a package for others to use, e.g. the Linux distributions. Specific issues are ensuring that the incoming licenses of component packages are compatible with each when combined, since some licenses have conditions which are incompatible with other license conditions.

3.4 End Users

End Users just use the application. The incoming licenses will usually not be of concern, since F/LOSS licenses are generally for the ongoing use of the code not the end user. No outgoing licenses should be needed.

3.5 Organisations

Organisations have different needs to end users. F/LOSS is no different from any software, where a business decision must be made about the benefits and constraints of any business artefact used, including the adoption of a software application or suite. A greater issue facing organisations is possibly the ongoing support and maintenance of their software [7], including the patching and possibly roadmaps of the software [8].

Dissemination (Organisations)

Some organisations wish to disseminate the source code. A Government might wish to have an authentication code distributed, and for developers and vendors to incorporated this into their software packages. One method of distribution would be to use a Non-Reciprocal type of license, which would allow any open source project to use the code, but would also allow closed source vendors to incorporate the code with no ongoing obligation. A Linking license might also be used, for greater control.

Internally Used without Development (Organisations)

Internal users of F/LOSS using it without development are possibly more concerned about support costs including training, patching and usability.

Internally Used with Development (Organisations)

For incoming code, with most F/LOSS code, an organisation should be able to take F/LOSS, use it, modify or adapt the source code to fulfil their own needs. If they do not distribute the application, they should not need to reveal the changes to the source code. Although this may differ with some obligation licenses. However there are valid reasons to reveal the changes, in that especially to get the benefits of the continual development of the program to obtain some control and certainly over the ongoing development, maintenance and direction of the codebase [9].

Externally Shared with Development (Organisations)

Some organisations, might share development, this might be part of the mission of the organisation, or to lower costs and risks. Where the organisation wishes to use the application with an external party the choice of incoming license is more important, depending on the outgoing considerations, or business requirements.

Outgoing licenses will depend on the outcomes desired by the organisation, perhaps using a reciprocal license to ensure that the code is open to all, or a membership or obligation license to enable the organisation to keep control over the software and code base.

4 Conclusion

This paper has described multiple types of stakeholders which now exist in the F/LOSS domain, and their different requirements of incoming and outgoing F/LOSS licenses. The purpose has not been to give a prescriptive directive to the matching of a license or type of license to a particular stakeholder but to try and give some background and definition as to the different types of stakeholders and what type of licenses match to their needs. Ultimately the choice of incoming and outgoing licenses should be a decision to fit with the aims and constraints of the individual project.

References

[1] L. Rosen, Open Source Licensing Software Freedom and Intellectual Property Law. Prentice Hall, 2004.
[2] D. Skidmore, "Free / Libre and Open Source Software: Describing Some Legal, and Software Engineering Terms, and a Taxonomy for Classifying Licenses," in Handbook of Research on Open Source Software: Technological, Economic, and Social Perspectives, K. St.Amant and B. Still, Eds. Idea Group, 2007, Chapter 31.
[3] J. Michaelson, "There's no such thing as a Free (software) lunch," ACM Queue, vol. 2, 2004.
[4] Apache Software Foundation, "Apache License, Version 2.0," 2004.
[5] Hacktivismo, "The Hacktivismo Enhanced-Source Software License Agreement," 2005.
[6] Squiz.net, "Squiz.Net Open Source Licence Agreement (Version 1.1),", 2005.
[7] D. L. Parnas, "Software Aging," Proceedings of the 16th international conference on Software engineering, Italy, 1994.
[8] S. Goode, "Something for nothing: management rejection of open source software in Australia's top firms," Information & Management, vol. 42, pp. 669-681, 2005.
[9] K. Edwards, "An economic perspective on software licenses—open source, maintainers and user-developers," Telematics and Informatics, vol. 22, pp. 97-110, 2005.

Part III

Panels

Introduction to Panel Discussions at the Third International Conference on Open Source Systems – OSS 2007

Sandra A. Slaughter[1]
1 David A. Tepper School of Business, Carnegie Mellon University, Pittsburgh, Pennsylvania, U.S.A. email: sandras@andrew.cmu.edu, web: http://www.tepper.cmu.edu/andrew/sandras

Summary

Two diverse, stimulating and important panel discussions are included in this year's programme on open source systems.

The first panel explores the diverse set of legal issues and risks that arise as governments increasingly adopt open source systems. Governments are attracted to open source software systems by the potential cost savings, open standards and protocols, and the flexibility, adaptability and reliability of the systems. However, there are serious legal issues and liabilities that can arise when governments adopt open source systems. This panel identifies and examines potential legal risks and liabilities from open source systems adoption in the government context. Some of the important legal concerns discussed by this panel include intellectual property rights, licensing, and protection of sensitive or confidential information.

The second panel is organized as a debate in which the panelists will consider the advantages and disadvantages of the sharing of research data and analyses by open source systems researchers. The area of open source systems has attracted numerous researchers from many different disciplines. In part, the attraction of research on open source systems is due to the public information available about open source communities. Using this information, researchers have been uncovering fascinating insights about how open source systems are developed and used and how open source communities function. Unfortunately, as noted by this panel, the work products of this research are not readily available to the public. This prevents new researchers from validating, building upon and extending the research already conducted by others. However, making research work products available to the others, while potentially valuable, has associated disadvantages. For example, certain kinds of data collected by a researcher (such as surveys or other confidential information) may not be released to the public, given human subjects concerns. In addition, it can require substantial cost and effort on the part of the researcher to make research work products understandable and usable by others. This panel debates the issues surrounding the pros and cons of making open source systems research work products available to researchers.

Legal issues for free and open source software in government

Nic Suzor[1], Brian Fitzgerald[1] and Mark Perry[2]
1 School of Law, Queensland University of Technology 2George St., Brisbane, Queensland Australia 4000
{n.suzor,bf.fitzgerald}@qut.edu.au,
WWW home page: http://www.oaklaw.qut.edu.au/
2 Faculty of Law, University of Western Ontario, 1151 Richmond, London, Ontario CANADA
mperry@uwo.ca,
WWW home page: http://www.csd.uwo.ca/~markp

As more governments begin to adopt and release free and open source software, it is important to be able to readily recognise and identify the associated legal risks and potential liabilities. This panel will examine and discuss the most common of these associated risks and liabilities. These issues include, but are not limited to:

- indemnities against claims of intellectual property infringement from third parties
- requirements of consumer protection and antitrust legislation
- obligations to redistribute source, and when they arise
- enforceability of free software licences
- layering and combining of licences
- dual-licensing
- licence incompatibility
- software patent liability
- contemporary developments in software licensing

There are significant advantages to a broad government adoption of free software. These range from potential cost savings, adoption of open standards and protocols, and wider use of stronger, more flexible and more secure software, to the social benefit derived from promoting a contributory commons of free software. However, governments ought to be aware of the obligations that may be imposed by the use and redistribution of FOSS, and when exactly these obligations will arise. Governments must also be mindful of the effect that implied warranties may have upon the sale or supply of free software, and the limitations inherent in indemnity clauses in many free software licences.

Where a government is using public funds to develop a software application, great care must be taken when choosing a licensing strategy. If there is a large commercial market for the unmodified application, a traditional closed source licensing approach can be considered to generate income. If the only commercial market for the software consists of software developers who would heavily modify or integrate the software, then a dual licensing approach could be taken to provide an income stream from those developers while still allowing the benefits of publicly-funded software to flow back to the community. Finally, where there is no commercial market for the software, or if the commercial market is more concerned with custom development and support services, there is a strong argument that the government should release the software under a free licence.

The evaluation of whether a government should use free or open source software for any given application is a complex matter. However, with the continual increase in quality and quantity of available solutions, coupled with increased understanding of the advantages and obligations involved, we can expect to see more widespread use of FOSS by governments across the world. In this context, the challenge for lawyers and government officials will be to fully understand the intricacies of this emerging area of law. This panel is but one step in gaining an appreciation of the legal landscape involved.

Data and analyses sharing to support research on free/libre open source software
A Debate

Brian Fitzgerald[1], Moderator

Proposition
Evangelia Berdou[2]
Kevin Crowston[3]
Greg Madey[4]

Opposition
Megan Conklin[5]
Stefan Koch[6]
Walt Scacchi[7]

1 University of Limerick, brian.fitzgerald@ul.ie
2 London School of Economics
E.Berdou@lse.ac.uk
3 Syracuse University
crowston@syr.edu
4 University of Notre Dame
oss@nd.edu
5 Elon University
mconklin@elon.edu
6 Wirtschaftsuniversität Wien
stefan.koch@wu-wien.ac.at>
7 University of California, Irvine
wscacchi@ics.uci.edu

Be it resolved, that the FLOSS research community requires that data and analyses behind FLOSS research publications be made expeditiously available to other researchers.

Research on FLOSS has relied on several different kinds of scientific evidence, such as the archives created by the FLOSS developers, versioned code repositories, mailing list messages and bug and issue tracking repositories [1]. FLOSS teams retain and make public archives of many of their activities as by-products of their open technology-supported collaboration. However, the easy availability of primary data provides a misleading picture of ease of conducting research on FLOSS. Precisely because these data are by-products, they are generally not in a form that is useful for researchers. Instead potentially useful data is locked up in HTML pages, CVS log files, text-only mailing list archives or dumps of website databases. FLOSS research projects, therefore, expend significant energy collecting and re-structuring these archives for their research, which is repetitive and wasteful [2]. Furthermore, different researchers will extract different data at different points in time, take different approaches to processing and cleaning data and make different decisions about analyses, but without all of these decisions being visible, auditable or reproducible. In principle, these latter problems can be addressed by individual researchers better documenting what they have done. However, research publications typically have restrictions on publication lengths that make complete discussion impossible. Furthermore, published papers are just the tip of the iceberg, and knowing what others have done does not necessarily make it any easier to replicate the results.

In light of these issues, FLOSS research might be greatly facilitated by increased sharing of primary data as well as various stages of data analysis. Such data archives have had some success in facilitating research in other fields, e.g., in biomedical sciences. However, there are numerous problems that must be addressed to make such data sharing feasible and valuable. One of the most important issues is ensuring the appropriate rewards and incentives for sharing. The experience in other fields suggests that if data sharing is an option, it is one that will be exercised by only a few researchers. On the other hand, it is not clear how sharing might be enforced or what the effects of such a mandate might be. The issue seems like one that is ready for a public debate.

Therefore, we will debate the resolution that the F/L/OSS research community requires that data and analyses behind F/L/OSS research publications be made expeditiously available to other researchers. If this resolution gains support from participants at the *International OSS Conference*, then efforts can be made to implement this resolution in the research field.

References

1. D. German and A. Mockus, in *Proceedings of the ICSE 3rd Workshop on Open Source*. (2003).
2. J. Howison, M. Conklin, and K. Crowston, FLOSSmole: A collaborative repository for FLOSS research data and analyses, *International Journal of Information Technology and Web Engineering* **1**(3), 17 (2006).

Part IV

Tutorials

Introduction to OSS 2007 Tutorial Program

Ernesto Damiani
DTI - University of Milan, Italy
damiani@dti.unimi.it

Today, most software researchers and professionals consider Open Source as a greatly successful paradigm for large-scale production of high-quality software systems. However, most of them would probably agree that successfully applying this paradigm requires dealing with a number of important issues. One of the best reasons to attend OSS conference series is being able to discuss these issues in an unprecedented gathering of top-notch researchers, developers, and leaders from all avenues of the open source movement. Attending good tutorials, in particular, is a great opportunity for getting in touch with new ideas and discussing them in depth with real experts.

For this reason, I am especially glad to introduce OSS 2007 tutorial program, which in my opinion does worthily complete the conference's rich program of research sessions and workshops. The aim of OSS tutorial series is to simply explain complicated subjects, attracting an audience whose composition will hopefully mirror the one of most free and open source software communities today, including absolute beginners as well as experienced open source gurus.

This year's rigorous selection process has selected an extraordinary set of tutorials which provide straightforward, but never trivial, introductions to a wide range of topics concerning the emerging role of open source in a number of application domains.

Namely, Megan Conklin's tutorial "How to Gather Metrics on FLOSS projects", will introduce the crucial problem of representing and gathering knowledge on FLOSS as a prerequisite to advanced decision making, e.g. on open source software adoption issues.

François Déchelle's tutorial "EDOS Tools for Linux Distributions Dependencies Management and Quality Assurance" will discuss a number of relevant issues related to testing and quality assurance of heterogeneous open source packages and present tools developed in the EDOS project for managing Linux distribution testing and quality assurance processes.

Finally, Hans-Ludwig Hausen's tutorial "Quality Specification, Testing and Certification of Bespoken, Open Source and Commercial Off-The-Shelf Systems" addresses some important software quality and metrology methods and procedures (specification, testing, V&V, reliability, safety, security, measurement, assessment,

Please use the following format when citing this chapter:

Damiani, E., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 359–360.

etc) suitable for dependable information system engineering and acceptance testing or for software certification.

The tutorial section in OSS 2007 Proceedings includes concise summaries of these three accepted tutorials, hopefully conveying some of the spirit of the lively exchange of ideas these tutorials will encourage at OSS 2007.

Putting together an attractive and scientifically sound tutorial program like this one is always a team effort. I would like to thank all OSS 2007 officials for their valuable comments and help in the selection process. Special thanks are also due to Alberto Colombo and Fulvio Frati, OSS 2007 Web masters, for their help in making the selection process as smooth and efficient as one might desire. Above all, I wish to thank all the tutorial proposers for choosing OSS 2007 as the preferred venue for presenting their ideas and techniques, and the tutorial attendees whose participation and feedback are essential for making OSS 2007 tutorial program a success.

How to Gather FLOSS Metrics

Megan Conklin[1], Jesus M. Gonzalez-Barahona[2], and Gregorio Robles[3]
1 Elon University, Department of Computing Sciences, Elon, NC 27244
mconklin@elon.edu,
WWW home page: http://facstaff.elon.edu/mconklin
2 Universidad Rey Juan Carlos, Grupo de Sistemas y Comunicaciones,
c\Tulipan s/n E-28933 Mostoles, Spain
{jgb,grex}@gsyc.escet.urjc.es,
WWW home page: http://libresoft.urjc.es/

Abstract. In this half-day tutorial, participants will gain hands-on exposure to key technologies for data collection about open source projects.

1 Program and Objectives

The tutorial will begin with reviews of the main source code repositories, including popular code forges such as Sourceforge, and techniques for collecting data directly from the forges as well as from aggregation projects such as FLOSSmole[1]. The tutorial will then discuss tools designed for analyzing the data found on forges, such as CVSAnalY[2], Pyternity, and SLOCCount, among others. Most importantly, participants will have a chance to analyze data with the help of the presenters. Teams of participants will solve open-ended analysis problems collaboratively and in real-time during the workshop. Finally, participants will have opportunities to discuss with the presenters what sort of data collection and analysis tools they would like to see built in the future.

Tutorial program:

- Briefly introduce overall problem of data collection
- Introduce tools: FLOSSmole, CVSAnalY, Pyternity, SLOCCount, etc
- Distribute data sets, pose problems for real-time assessment
- Share results
- Discuss future prospects

[1] http://ossmole.sf.net
[2] http://cvsanaly.tigris.org/

2 Background of Presenters

Megan Conklin is an assistant professor in the Department of Computing Sciences at Elon University. Her primary research focus is on data mining and large database systems, particularly for software engineering data. She was co-organizer of the 2006 WoPDaSD workshop at the International Conference on Software Engineering (along with Gregorio Robles and Jesus Gonzalez-Barahona). She has published a number of papers on tools for analyzing open source projects, and has spoken about open source data collection at such diverse events as the Mining Software Repositories workshop at ICSE and the O'Reilly Open Source Convention. She has a PhD in computer science from Nova Southeastern University.

Jesus M. Gonzalez-Barahona teaches and researches at Universidad Rey Juan Carlos, Mostoles (Spain). His research interests include libre software engineering, and in particular quantitative measures of libre software development and distributed tools for collaboration in libre software projects. In this area, he has published several papers, and is participating in some international research projects (more info at http://libresoft.urjc.es). He is also one of the promoters of the idea of a European masters program on libre software, and has specific interest in education relating to that area.

Gregorio Robles is Associate Professor at the Universidad Rey Juan Carlos in Madrid, Spain. He earned a degree in electrical engineering from the Universidad Politécnica de Madrid (studying his last year and submitting his master thesis at the Technical University of Berlin, DE) and obtained his PhD in 2006. His research work is centered in the study of libre software development from an engineering point of view, especially with regard to quantitative and empirical issues. Related, non-technical matters have also been of interest: volunteer-driven software development and social network analyses of the libre software phenomenon. He has developed or collaborated in the design of programmes to automate the analysis of libre software and the tools used to produce them. He was also involved in the FLOSS study on libre software financed by the European Commission IST programme, and was involved in other European-funded projects such as CALIBRE or FLOSSWorld. He has also had the opportunity to attend the following universities as a research visitor: Wirtschaftsuniversität Wien (AT, 2 months), MERIT/University of Maastricht (NL, 4 months), the University of Lincoln (UK, 3 months) and the Technical University Munich (DE, 5 months).

EDOS-Tools Tutorial: EDOS Tools for Linux Distributions Dependencies Management and Quality Assurance

François Déchelle[1], Fabio Mancinelli[2]
[1]EDGE-IT, France
fdechelle@mandriva.fr
[2]PPS - Université Paris VII, France
fabio@pps.jussieu.fr

Abstract. Free and Open Source Software (FOSS) distributions are the results of the effort of third party actors in collecting independently developed software products, in a consistent and usable form. The widespread adoption of these distributions as infrastructural components in many strategic contexts of the information technology society has drawn the attention on the issues regarding how to handle the complexity of assembling and managing a huge number of (packaged) components and how to guarantee their quality. This tutorial will describe how the EDOS project has tackled these issues. First it will describe the problems related to the quality assurance of Linux distributions and will present the tools that have been developed to manage testing process. It will then introduce the problems that occur when managing inter-package relations in large package repositories and will showcase tools that can be used to analyze and manage large package repositories.

Description

The tutorial will be organized in two parts. The first part will introduce the issues related to testing and quality assurance of heterogeneous Open Source packages and present tools developed in the EDOS project for managing Linux distribution testing and quality assurance processes. The second part will introduce the state of the art in Linux package management systems and problems regarding the management of inter-package relations (dependencies and conflicts) in large package repositories.

1.1 Testing and quality assurance

The tutorial will detail the use of the following tools:

- **Testrunner**: a tool for conducting automatic and manual tests and reporting test results. Testrunner uses an XML-based test specification and can report test results using several reporting plug-ins, for instance to report results to the QA portal using HTTP request.
- **TULIP**: a tool to test upgrades of Linux installations using virtual machines and the distribution standard upgrade tools. TULIP can run automatic upgrades of installed Linux distributions, test the upgraded distributions and reports results to the QA portal using Testrunner.
- **QA Portal**: a web portal for test management, that allows testers and distribution developers to have a real-time and accurate view of the distribution testing process including available test suites, tests, reports of executed tests...

The tutorial will present how to install the tools, how to setup a complete distribution testing environment and will feature a hands-on session on a realworld distribution testing process.

1.2 Large package repositories complexity and dependency management

The second part will introduce the state of the art in Linux package management systems and problems regarding the management of inter-package relations (dependencies and conflicts) in large package repositories. A set of tool that can be used by distribution editors to analyze and manipulate repositories in order to find potential problems due to incorrect inter-package relation specifications will then be showcased:

- **DEB/RPMCheck**: a dependencies correctness checker. DEB/RPMCheck provides a fast way for analyzing whole package repositories and to spot problems that can be present in package dependency meta-data.
- **History**: historical analysis and symbolic manipulation of package repositories. History is powered by a powerful functional language called DQL that enables the user to perform sophisticated queries on package repositories and to manipulate them in a declarative way by using some advanced operators. Moreover, History supports the analysis of historical data for tracking the evolution of package repositories over time.
- **Anla**: a web service for package repository exploration. Anla is the web-oriented counterpart of History that can be used by distribution editors to provide a direct feedback on the distribution status to users, testers and developers. Advanced queries can be performed using this interface and hyperlinked graphical results are provided as output.
- **Tart**: an optimized “thinner” for building custom distributions. Tart enables distribution editors to build custom distributions that met some constraints (e.g. space or priority). By using Tart it is possible to create package sets that are closed with respect to dependency relations and that satisfy the optimization needs defined by the constraints.

Quality Specification, Testing and Certification of Bespoken, Open Source and Commercial Off-The-Shelf Systems

Hans-Ludwig Hausen
FRAUNHOFER
Schloss Birlinghoven, D-53754 St. Augustin, Germany
Hans-Ludwig_Hausen @_ fit.fraunhofer.de

Abstract. The seminar will cover the principles as well as the best practices of software system quality assurance (comprising inspection, verification, validation, black and white box test, measurement and assessment, and the normative quality characteristics) for procedural, object-oriented, aspect-oriented and agent-based dependable software. Attendees will exercise proven techniques for goal-directed quality specification, testing, measurement, scaling and assessment for software certification. Assessment of both the software product as well as the software process will be discussed with respect to its relevance for such acceptance assessments. A standardized process model for measurement, assessment and certification of dependable software will be used to make the attendees familiar with this comprehensive assessment procedure and to learn how to embed it into today's standardized or non-standardized software processes..

What is software quality and what is quality specification, evaluation, assessment and certification? Why do we need it? Well, we ask for software quality specification, assessment and certification because we want to be sure that the product we want to apply provides the expected service correctly with respect to both functional and non-functional requirements. If we are assessing software we check whether the actual service we can get from the present version of the product is (at least to some degree) equivalent to the required service. We assume that the actual service is provided by a program that has been coded under several conditions and constrains and thus not being a one-to-one translation of the required service. Complementary we introduce a third layer in our approach called specified service, where we define what has to be done on the computer. The actual service might be considered as the layer describing how the service is to be accomplished. Why and what for the service is needed is already defined in the required service layer.

In such a layered product environment assessment is performed using assessment methods such as inspection, testing, verification and measurement to check the actual service against the specified service and the expected service. These assessment methods have to be supported by appropriate tools. For the assessment we also need to know which characteristics of the product have to be considered and what is the threshold for them. In order to be able (i.e. "to be allowed") to certify a product, i.e. to put a quality seal on the product, we have to evaluate assess all product layers with the required characteristics using appropriate methods and tools on both product documents and process documents. As a consequence we have to handle product, process, characteristics, methods and tools as wells as their interaction in a defined, coherent procedure.

For effective quality specification, testing and certification the product and process elements have to be identified and evaluated with respect to selected, required characteristics. Appropriate methods and tools have to be applied to the product and process documents to check those characteristics. The essential problem domains are: software product, software process, software characteristics, software methods and software tools for procedural, object-oriented or agent-based dependable software systems. Proven techniques for goal-directed quality specification, measurement, scaling and assessment are mandatory. Obviously one has also to consider norms, regulations or standards such as the ones for

- Software Quality Specification and Evaluation: ISO9126 and ISO 14598
- Evaluators Guide according ISO9126 and ISO 14598
- COTS Quality Specification and Evaluation: ISO 12119
- COTS Evaluation Guide according ISO 12119
- ISO 25000 series

And finally, a standardized process model for measurement, assessment and certification of dependable software is required applicable in the context of today's standardized as well as within non-standardized software processes.

References

1 Software Evaluation for Certification; Andrew Rae, P. Robert, Hans-Ludwig Hausen; McGraw-Hill, Inc. New York, NY, USA, (new version in progress) c.f.: http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=561101

2 A method for software evaluation; Dieter Welzel, Hans-Ludwig Hausen; Computer Standards & Interfaces; Volume 17 , Issue 1, Pages: 121 – 129; c.f.: http://portal.acm.org/citation.cfm?id=198664

3 Guides to Software Evaluation for Acceptance Testing; Hans-Ludwig Hausen, Internal Workbook, (available from the author)

Part V

Workshops

Introduction to Workshops at the Third International Conference on Open Source Systems – OSS 2007

Scott A. Hissam[1]

1 Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, U.S.A. email: shissam@sei.cmu.edu, web: http://www.sei.cmu.edu/staff/shissam

The International Conference on Open Source Systems owes its success to the increasing desire of researchers and academicians to engage in open conversation and sharing with the open source community regarding its' motivations, products, processes and data. This diverse representation of the growing practical and academic interest in Open Source Systems and its impact on the software engineering community is represented in the five workshops that accompany this year's conference. OSS 2007 welcomes these workshops as participants engage, in an informal setting, driven by fundamental academic interests, or by more applied industrial or commercial interests to discuss technical issues, exchange research ideas, and to develop a community in the field of Open Source Software and/or Systems.

The 1st International Workshop on Trust in Open Source Software (TOSS) explores many issues that effect trust in the use and adoption of open source products including legal aspects, quality, and business models.

Open Source Software and Product Lines 2007 aims to examine the mechanisms used in Open Source Software production and software product lines as a means to develop a greater understanding how each of these communities of developers can be of mutual benefit.

OSS in Economic and Managerial Perspective wishes to expand economic and managerial research agendas by engaging researchers in Open Source Software Engineering and assessing any potential gaps between where those economic and managerial researchers are in relation to the current state of Open Source Systems.

The 2nd Workshop on Public Data about Software Development seeks to foster the analysis of publicly available regarding Open Software Development and the exchange of these data and analyzes between different researchers.

The Workshop on Free and Open Source Learning Environments and Tools provides a discussion forum for researchers and practitioners in the use of free and Open Source Software and Applications in Web-based Learning Environments in an aim to identify effective and sustainable solutions, open issues and good practices.

1st International Workshop on Trust in Open Source Software (TOSS)

Sandro Morasca[1], and Alberto Sillitti[2]
1 Università dell'Insubria, Como, Italy, sandro.morasca@uninsubria.it
2 Libera Università di Bolzano, Bolzano, Italy, alberto.sillitti@unibz.it

Abstract. The 1st International Workshop on Trust in Open Source Software (TOSS) focuses on one of the major factors in the adoption of software solutions: the trustworthiness of OSS, which has influenced the widespread adoption of OSS in industry. Software quality aspects have been investigated for a long time in the academia but the usage/production of OSS is creating new challenges and the points of view of the industry and the academia may differ, especially on the trustworthiness of OSS. The aim of the workshop is to open a communication channel between the industry and the academia focusing on this issue and promote a long lasting discussion on it. The workshop tries to identify the different points of views and the different approaches that can result in benefits for the software industry.

1 Introduction

Open Source Software (OSS) is supported by the major players in the software industry. However, software companies are still somewhat reluctant to adopt OSS massively in their mainstream development, mainly because of lack of trust regarding OSS. There are several reasons that affect trust, including legal aspects, quality, and business models.

Trust is not an attribute that can be claimed without backing evidence. It also relies on perception, on answers to non-technical questions such as "who is behind Open Source?", "why be confident in OSS?", or even "how to be confident in OSS?" If OSS aims to be considered as good as proprietary software, these questions must be addressed.

Using OSS is not an easy choice for companies, since the selection process is affected by several factors related to the quality of the source code and the quality of the production process.

OSS is often developed not only by individuals, but by communities and companies that follow a rigorous process and high quality standards which they use

Please use the following format when citing this chapter:

Morasca, S. and Sillitti, A., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 371–373.

in the development of commercial products (e.g., there are several commercial products that have been released as OSS). However, verifying the quality of OSS products is complex since there are almost no rules for evaluating and describing it in a way that different companies/communities can trust. A difficult challenge is the development of a set of guidelines/tools/processes that companies can follow to produce and to adopt OSS that also other companies/communities can trust.

Several EU-funded projects address such problems in OSS from different points of view and in different domain areas. The organizers of this workshop and the member of the industrial board are partners in QualiPSo, the largest project funded in this area.

Several EU-funded projects address such problems in OSS from different points of view and in different domain areas. The organizers of this workshop and the some members of its Industrial Board are partners in QualiPSo, the largest project funded by the EU in this area.

2 Organization of the workshop

The workshop will include three main sections:

1. Presentation of the industrial needs
2. Presentation of the research activities in the area
3. Open discussion guided by the Industrial Board

The first section will include presentations from industry identifying the main challenges and objectives for large, medium, and small companies that use and produce OSS not only as product but also as part of other products (e.g., embedded systems).

The second section will include presentation from the academia identifying the areas and the results reached in the area from the technical/legal/business point of view.

The third section will be an open discussion guided by the Industrial Board in which the presentations of the first two sections will be discussed.

3 Duration

The duration of the workshop is one full day.

4 Contributions

Participants are expected to submit a short paper or a position paper (4 pages max.) to the conference organizers.

5 Aims and objectives

The main objectives of the workshop are the following:

- Group together people from industry and the academia interested in trust and quality
- Enable knowledge transfer between academia and industry
- Allow people from industry to highlight the main problems in the area
- Discuss the elements of trust in different application domains

6 Participants

The workshop focuses on the industrial aspects of the research in quality applied to the Open Source Software, therefore the intended participants will include people from both academia and industry.

7 Program Chairs

Sandro Morasca	Università dell'Insubria	Italy
Alberto Sillitti	Libera Università di Bolzano	Italy

8 Industrial Board

Stefano De Panfilis	Engineering Ingegneria Informatica	Italy
Jean-Pierre Laisne	Bull	France
Stéphane Laurière	Mandriva	France
Gregory Lopez	Thales	France
Clara Pezuela	Atos-Origin	Spain

Open Source Software and Product Lines 2007

Workshop at Third International Conference on Open Source Systems

Frank van der Linden, Björn Lundell
Philips Medical Systems, University of Skövde

Workshop Topics

Embedded industries have invested a lot in the introduction of software product lines in their software development. In addition, using open source software appears to be a profitable way to obtain good software. This is also applicable for organizations doing product line engineering. On the other hand, because of the diverse use of open source software, product line development is an attractive way of working in open source communities. In fact, the configuration mechanisms used in open source communities may be applicable within software product lines as well. In addition, product line organisations are usually involved in distributed development, which works very efficiently within open source communities. However, at present, there is limited interaction between the open source and product line development communities. The aim for the workshop is to explore what the two communities can learn from each other and to develop a better understanding of how the two communities can benefit from each other. The workshop deals with the following issues:

- Community: Ownership, control and management of product line assets in an open source community
- Visibility of the code: when it is valuable to share proprietary code and how to take the right decision.
- Architecture Views: Creation of different levels of architecture visibility: proprietary, among closed consortium, public. Is this possible?
- Product line requirements roadmaps and planning in open source development
- Variability management: Using the open source community to evolve components and being explicit about variability
- Variability representation: in an open source community
- Deployment: Open source for the platform and in applications

- Heterogeneous processes: Cohabitation of product line management and agile processes
- Tools: Open source asset management tools in product line development
- Domain and application engineering and their meaning in an open source context
- Recovery and recognition of a product line in an open source asset base
- Legal: Aspects dealing with evolutionary, variability or distribution of development relating to legal risks involving: liability, warranties, patent infringements etc.

Previous workshop

A previous workshop OSSPL06 with the same topic was held during the SPLC 2006 conference in Baltimore. Proceedings are at the OSSPL06. The OSSPL06 workshop resulted in the following topics that need further investigation:

- Human issues: The way that people are recruited and fired, Culture
- Tools: OSS has simple useful tools - they integrate! Especially there are good OS asset management tools. Within OS, variation is often implicit, embedded in packages
- Processes & maturity: Are all SPL practices necessary?
- Organisational issues: Ownership, fear of the unknown by commercial organisations
- Architecture: SPL concentrates on models, OSS on code/configurations. A further aspect of investigation is the assessment of open and close source quality
- Business: How/when to use OSS in SPL - what about participation?

Program committee:

Frank van der Linden, Philips Medical Systems
Josetxo Vicedo, European Software Institute
Pentti Marttiin, Nokia
Hans Petter Dahle, ICT-Norway
Jesús Bermejo, Telvent
Björn Lundell, University of Skövde

Toward a New Industrial Organization? OSS in Economic and Managerial Perspective

Jean Michel Dalle[1], Cristina Rossi[2], and Francesco Rullani[3]
1 Universit´e Pierre et Marie Curie, Paris, France; jean-michel.dalle@upmc.fr,
2 Politecniico di Milano, Department of Management, Economics, and Industrial Engineering
P.zza Leonardo da Vinci 32, 20133, Milano, Italy
{cristina1.rossi}@polimi.it,
3 Copenhagen Business School (fr.ivs@cbs.dk)

Abstract. At present, an more and more users are running Open Source software (OSS) on their systems. Major companies, like IBM, Oracle, or Sun Microsystems, have now started to make significant investments in developing open communities and creating a portfolio of systems incorporating OSS applications into their design. Meanwhile, an increasing number of firms are entering the market by offering OSS-based solutions to their customers, often supplying a mix of proprietary and open solutions through hybrid business models. In this context, economists and management scientists are now moving beyond the state of puzzlement that has driven much of the initial attention towards OSS. Located in the context of OSS2007 in order to foster close and fruitful interactions with scholars from various other disciplines, this workshop aims at contributing to the current evolutions of the economic and managerial research agendas about OSS, and thus to provide, first, an assessment of where we - economics and management scholars - are about OSS, and, second, an analysis of the renewed directions in which we should consider inquiring further in the near future, focusing notably on business, production, diffusion and innovation models.

1 Introduction

In December 2006, Apache had a market share of 60.64% against 30.67% for its immediate competitor. Linux has already been adopted by approximately 29 millions of users. The market share of the Web browser Firefox is surging despite the pre-existing dominance of Internet Explorer*. Major companies like Oracle have now started providing professional support on Linux. IBM, which had been promoting Linux offerings for years, has strengthened its commitment to openness, making

Please use the following format when citing this chapter:

Dalle, J.M., Rossi, C. and Rullani, F., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 377–379.

significant investments in the development of communities and creating a broad portfolio of systems adopting open standards and incorporating open-source applications into their design. Sun's Java is open-source. And meanwhile, an increasing number of firms are entering the market by offering open-source-based solutions to their customers, often supplying a mix of proprietary and open solutions through hybrid business models.

It is time to acknowledge the fact that Open-Source software now belongs to the mainstream of the software industry, and that it is rapidly modifying major elements of its industrial organization.

In this context, economists and management scientists are now moving beyond the state of puzzlement that has driven much of the initial attention towards open-source software and related systems.

This workshop aims at contributing to the current evolutions of the economic and managerial research agendas about open-source software providing, first, an assessment of where we - economics and management scholars - are about OSS, and, second, an analysis of the renewed directions in which we should consider inquiring further in the near future, focusing notably on business, production, diffusion and innovation models.

For this reasons we thought about the OSS2007 conference as a perfect context to develop a close and fruitful discussion around these topics. If on the one hand the perspective of the workshop will be able to attract a large number of economists and managerial scholars, on the other hand the collocation at the OSS2007 conference will foster the participation of scholars from various other disciplines, and notably software engineering researchers, creating an interdisciplinary milieu and enriching the debate.

2 List of topics

The following list of topics displays a non-exhaustive sample of the possible themes:

1. *Business models*: OSS strategies of large and small software firms, sustainability of OSS-based business models, hybridization between commercial and free/open software, relationships between software firms and OSS communities (firms' contributions and strategies, role of so-called paid developers, role of networks, etc.), implications of OSS licensing, and of copyright and patent issues, etc.
2. *Production models*: specificities of the OSS production model and their economic and managerial consequences, allocation and coordination mechanisms, characteristics of projects and of project ecologies, modularity issues, etc.
3. *Diffusion models*: empirical evidence on OSS diffusion, typologies of users and of adoption motives, users' characteristics, role of users' communities, market dynamics of OSS and proprietary software, integration and competition of open and closed solutions, etc.

4. *Innovation models*: open innovation, factors favouring the emergence of OSS-like innovation regimes, policies dedicated to the software and to other sectors, open-source technology transfer, etc.
5. *Standardization and platform strategies*.

3 Organisation and scientific committee

The **Organization Committee** is formed by
Jean-Michel Dalle, Université Pierre et Marie Curie (jean-michel.dalle@upmc.fr)
Cristina Rossi, Politecnico di Milano (cristina1.rossi@polimi.it)
Francesco Rullani, Copenhagen Business School (fr.ivs@cbs.dk)

The members of the **Scientific Committee** are:
Cristiano Antonelli, Professor, University of Turin, IT
Stefano Comino, Assistant Professor, University of Trento, IT
Linus Dahlander, Research Fellow, Imperial College London, UK
Jean-Michel Dalle, Professor, Pierre-et-Marie-Curie University, FR
Paul David, Professor, Stanford University, US; Oxford Internet Institute, UK
Lars Frederiksen, Research Associate, Imperial College London, UK
Alfonso Gambardella, Professor, Bocconi University, IT
Marco Giarratana, Assistant Professor, Universidad Carlos III, ES
Lars Bo Jeppesen, Assistant Professor, Copenhagen Business School, DK
Joachim Henkel, Professor, Technical University of Munich, GE
Hely Koski, Professor, Helsinki School of Economics, FI
Yuwei Lin, Research Associate, University of Manchester, UK
Alessandro Nuvolari, Research Fellow, ECIS, Eindhoven University, NL
Margherita Pagani, Adjunct Professor of Management, Bocconi University, IT
Lucia Piscitello, Professor, Polytechnic of Milan, IT
Alessandro Rossi, Assistant Professor, University of Trento, IT
Cristina Rossi, Assistant Professor, Polytechnic of Milan, IT
Francesco Rullani, Post-doc Fellow, Copenhagen Business School, DK
Philipp J.H. Schroeder, Assistant Professor, Aarhus School of Business, DK
Sonali Shah, Assistant Professor, University of Illinois at Urbana-Champaign, US
Dominique Torre, Professor, University Nice Sophia Antipolis, FR
Salvatore Torrisi, Associate Professor, University of Bologna, IT

4 Program

The workshop will be help of June 14th 2007. The program provides for four thematic section, in which three papers will be presented. Each paper will be commented by a discussant.

Professor Paul A. David (Stanford University and Oxford Internet Institute) will give the introductory talk to the workshop on the topic: "Social Science Research Approaches to FLOSS".

2nd International Workshop on Public Data about Software Development (WoPDaSD 2007)

Jesus M. Gonzalez-Barahona, Megan Conklin, Gregorio Robles
http://libresoft.urjc.es/Activities/WoPDaSD2007

Abstract. Exchange of detailed data about software development between research teams, and specifically about data available from public repositories of libre (free, open source) software projects is becoming more and more common. This workshop will explore the benefits and problems of such exchange, and the steps needed to foster it. As a case example of data exchange, the workshop organizers suggest two large datasets to be analyzed by participants.

Introduction

In the latest years, and specially thanks to the huge availability of data about software development that can be obtained from libre (free, open source) projects, the research community is starting to produce, use and exchange large data sets of information. These data sets have to be retrieved, purged, described, and can be published for public consumption by other groups. Their availability allows for the decoupling of research activities (some groups can focus on data retrieval and preliminary analysis, which others can devote to more in-depth analysis without bothering with data retrieval), the reproducibility of research results, and even the collaboration (and competition) in the analysis of data.

All this activity is being presented in several workshops and conferences, but a single place to exchange experiences does not exist yet. We propose this workshop as such a place, where researchers in the field can discuss specifically about this kind of data sets, how they are retrieved, how can they be analyzed and mined, how they can be exchanged and complemented, etc.

Please use the following format when citing this chapter:

Gonzalez-Barahona, J.M., Conklin, M. and Robles, G., 2007, in IFIP International Federation for Information Processing, Volume 234, Open Source Development, Adoption and Innovation, eds. J. Feller, Fitzgerald, B., Scacchi, W., Sillitti, A., (Boston: Springer), pp. 381–383.

Main Goals

The goal of this workshop is to foster the analysis of public available data sources about software development and the exchange of data between different research groups.

The workshop is aimed specifically at two different target studies:

- Analysis of some data collections about software development (provided by the organizers, see below). The analysis should show a methodology for exploring any of those data sets (or better, to relate both) searching for some specific result in the area of software development, and its applications to the actual data sets. The study can be in the field of software engineering, economics, sociology, human resources, and others.
- Retrieval process and exchange formats of public available data collections about software development. The data collections presented should be publicly available, based themselves on public data (so that other groups could reproduce the data collection process), and be related to the field of software development. This includes, but is not limited to, data from source control systems, but tracking systems, mailing lists, websites, source and binary code, quality assurance systems, etc. Although any kind of data collection can be considered, those including information about a large amount of projects will be considered especially appropriate.

The target audience is composed by the research groups interested in empirical software engineering and quantitative studies of the software development processes and methods. This includes not only software engineers, but also researchers from other fields that might use the data for economic, social and other studies.

Detailed Description

Following the goals described above, the workshop will accept papers about two specific issues:

- Analysis of two data collections about libre software development: FLOSSMole and CVSAnaly-SF. These collections, already available to any researcher, are offered for the analysis. The studies submitted should detail how they have been used, which part of the information has been considered, how they have been validated or filtered and/or post-processed (if that is the case). The description should be detailed enough to let any other research group reproduce the study.
- Studies about the data retrieval and preparation for public consumption of data sets in the same realm, which could be proposed for analysis in future editions of the workshop.

FLOSSMole

FLOSSMole (formerly OSSmole) is a set of tools for gathering data (metrics) about the development of free/libre/open source projects. The FLOSSMole project also publishes the resulting analysis about FLOSS projects, and accepts data donations from other research groups. It offers this workshop a complete set of data gathered from the SourceForge development platform and the Freshmeat announcement systems. More information can be obtained from http://ossmole.sourceforge.net.

CVSAnalySF

CVSAnalY is a tool created by the Libre Software Engineering Group at the Universidad Rey Juan Carlos that extracts statistical information out of CVS (and recently Subversion) repository logs and transforms it in database SQL formats. It has been used to retrieve information for all projects that have an active CVS system at SourceForge. This data set is publicly offered to be analyzed in this workshop. More information can be obtained from http://libresoft.urjc.es/Data.

Challenge

This edition an specific challenge is proposed to contributors, in addition to regular papers. The topic of the challenge is "data visualization", and will consist on papers about visualization of the data in any of the datasets offered (FLOSSMole, CVSAnaly-SF, or both). The text in the paper should explain the visualization technique used, and its possible applications. The images in the paper should be the visualization images themselves, or snapshots of them. Visualization techniques that help to answer interesting questions, to better understand the data, or to find relationships in it (including relating data in both datasets) are encouraged.

FOSLET 07 – Workshop on Free and Open Source Learning Environments and Tools

Luca Botturi, Riccardo Mazza, Stefano Tardini
University of Lugano, eLab – eLearning Lab
via Buffi 13, 6900 Lugano, Switzerland
{botturil, mazzar, tardinis}@lu.unisi.ch
http://www.elearninglab.org

Introduction

Web-based Learning Environments supported by Course Management Systems (also known as Learning Management Systems) are nowadays a valid solution for institutions, companies, schools and universities that deliver eLearning or support blended-learning activities. Learning Environments are used to distribute information and content material to learners, prepare and deliver assignments and tests, engage in discussions, and manage distance classes without time and space restrictions.

During the last few years, several institutions have moved from commercial/proprietary solutions to Free and Open Source (FOS) environments. The increasing popularity FOS solutions for eLearning are enjoying is partly due to the absence of license costs, and partly to their great adaptability and interoperability, also in relationship with the development and adoption of Learning Technology Standards, such as the Shareable Content Object Reference Model (SCORM) and the IMS Content Packaging and Learning Design specifications.

However, the integration of FOS solutions for e-learning is not free from some critical issues that demand research.

Why FOS in eLearning?

What are the main reasons that may push institutions, companies, schools and universities to adopt FOS solutions for their e- and blended learning activities? The perceived benefits of FOS solutions usually concern three aspects, namely costs, infrastructure, and tailoring and integration.

Costs. The issue of costs is particularly relevant when dealing with the choice of an LMS: as a matter of fact, one of the main issues with commercial LMS is funding. The uncertain benefits of online learning may lead an institution to doubt about the real return of a huge investment as the acquisition of the required number of seats in a commercial LMS. First, instructors and students may not have established practices in using online tools, so that the actual use of the LMS is unpredictable. Second, the uncertainties of the market and the rapid and often earthshaking developments of the eLearning world may make the commitment to a single producer tricky. Finally, being committed to a commercial LMS risks being a one-shot situation: in the undesired chance of a failure, costs may make almost impossible to try out another solution. Choosing an FOS solution mitigates these three issues. Furthermore, the (almost) complete visibility of the life of an FOS community provides more information about its hope of survival in the eLearning market than the financial reports of super-protected commercial players.

Infrastructure (material and human resources). One of the big issues of FOS software, and one of its major hidden costs, is the need for infrastructure (hardware and network connection) and of in-house work for setting up the system, for maintaining the application and for checking, selecting and installing updates. All of these issues are quite unproblematic in most universities, since they (always) have a dedicated IT staff able to care after the infrastructure, the installation, maintenance and update of software applications. Moreover, the hardware demands of OS software are usually significantly lower than those of commercial software.

Tailoring and integration. An eLearning system potentially impacts the core of a university's activity, and has to be integrated with standard procedures for class scheduling, enrollments, assessment, quality evaluations, network accounting etc. The main advantage that an FOS solution brings to institutional users is the possibility to tailor the application to one's needs, and to integrate it in first person in existing procedures and IT system [2: 125].

FOSLET 07 – The workshop

In this context, the idea of promoting a workshop on FOS solutions for e- and blended learning activities arose from the experience gained by the authors in two labs of the University of Lugano (USI): the NewMinE Lab (New Media in Education Laboratory: www.newmine.org) and the eLab (eLearning Lab: www.elearninglab.org). In 2004 an OS LMS was introduced in the University of Lugano and in the University of Applied Sciences of Southern Switzerland (SUPSI) in order to support the educational activities and to promote the use of eLearning in the teaching and learning practices of both institutions [1, 2]. Then, in 2006 an OS Learning Objects Repository was developed by eLab and integrated into the LMS (DOOR – Digital Open Objects Repository: http://door.elearninglab.org).

The introduction of e-Courses (this is the name of the platform introduced, based on the Moodle technology; see http://corsi.elearninglab.org) soon raised interesting issues concerning the installation of the platform, its integration into the different existing universities' systems, its customization according to both institutions' needs, the promotion of the new platform among faculty members and students of USI and SUPSI and its evaluation, which showed high satisfaction levels. Hence the interest for other similar experiences and the chance of promoting within the Second

International Conference on Open Source Systems (OSS 2006) the first FOSLET workshop [3].

FOSLET 07 aims at providing a discussion platform for researchers and practitioners who want to share research findings and experiences about the use of FOS applications for eLearning, in order to identify effective and sustainable solutions, open issues and good practices.

The workshop will focus on solutions, practices, and experience of FOS Learning Environments that give particular emphasis to three perspectives:

1. *Interoperability*, including course content migration, metadata, and standards implementation.
2. *Integration*, including LMS support to specific teaching and learning activities, fine-tuning to specific organizational and administrative requirements.
3. *Sustainability*, including studies about the real cost of software, management issues, long-term financing and reusability.

Specific topics include: interoperability and implementation of standards among Learning Environments; FOS Learning Environments architectures, configurations, and infrastructures; Learning & Content Management Systems, repositories; metadata models and mapping; interoperability use cases, experience reports of interoperable educational systems; cost models of commercial/proprietary and FOS e-learning solutions; good practices of integration of FOS solutions into real institutional settings.

References

[1] Botturi, L. Functional Assessment of Some Open Source LMS. eLab Report, Lugano, November 2004. http://www.elearninglab.org/docs/risorse/report/OS_review_Nov2004.pdf.
[2] Botturi, L., Cantoni, L. and Tardini, S. Introducing a Moodle LMS in Higher Education: the e-Courses Experience in Ticino (Switzerland). Je-LKS. Journal of e-Learning and Knowledge Society, Special Issue: C. Giovannella (ed.), Emerging learning environments and Open source in e-learning, 2 (1), March 2006, 123-130.
[3] Botturi, L., Mazza, R. and Tardini, S. (eds). FOSLET 2006. Proceedings of the Workshop on Free and Open Source Learning Environments and Tools. University of Lugano – NewMinE ePaper 6, Lugano 2006. http://www.elab.usilu.net/foslet06/proceedings/NewMinE_ePaper6.pdf.

Part VI

Posters

List of Posters Displayed at The Third International Conference on Open Source Systems

Heterogeneous collaborative development involving open and inner source: Challenges for the European Software Intensive Industry

- Frank van der Linden (Philips Medical Systems, Best, The Netherlands, frank.van.der.linden@philips.com)
- Björn Lundell (University of Skövde, P.O. Box 408, SE-541 28 SKÖVDE, Sweden, bjorn.lundell@his.se)
- Pentti Marttiin (Nokia, Finland, pentti.marttiin@nokia.com)

Taking advantage of Open Source benefits for boosting growth in industry

- Clara Pezuela (Atos Origin, Spain, clara.pezuela@atosorigin.com)
- Gregory Lopez (Thales, France, gregory.lopez@thalesgroup.com)

Sound tools for package dependency management in Free and Open Source Software distributions

- Fabio Mancinelli (Université Paris VII, fabio@pps.jussieu.fr)
- Roberto di Cosmo (Université Paris VII, dicosmo@pps.jussieu.fr)
- Jérôme Vouillon (Université Paris VII, vouillon@pps.jussieu.fr)
- Jaap Boender (Université Paris VII, boender@pps.jussieu.fr)
- Berke Durak (INRIA Rocquencourt, berke.durak@inria.fr)
- Xavier Leroy (INRIA Rocquencourt, xavier.leroy@inria.fr)
- Ralf Treinen (LSV, ENS de Cachan, CNRS UMR 8643, INRIA Futurs, treinen@lsv.ens-cachan.fr)

Openware Integration Technique for In-house Software and Open Source Components

- Janne Merilinna (VTT Technical Research Centre of Finland)
- Mari Matinlassi (VTT Technical Research Centre of Finland)

Generating and Visualising Organisational Structures of Free/Libre and Open Source Software Projects

- Ludger Bischofs (OFFIS, Escherweg 2, 26121 Oldenburg, Germany ludger.bischofs@offis.de)
- Wilhelm Hasselbring (University of Oldenburg, FK-2, Software Engineering Group, PO Box 2503, 26111 Oldenburg, Germany hasselbring@informatik.uni-oldenburg.de)

OSS design science and its influence on OSS effectiveness

- Nassim Belbaly (GSCM, France)
- Hind Benbya (GSCM, France)
- Régis Meissonier (GSCM, France)

FLOSS as Democratic Principle

- Mark Perry (Faculty of Law, University of Western Ontario)
- Brian Fitzgerald (School of Law, Queensland University of Technology)
- Nic Suzor (School of Law, Queensland University of Technology)

Elements of Open Source Community Sustainability

- Niklas Vainio (University of Tampere)
- Ville Oksanen (Helsinki University of Technology)
- Tere Vadén (University of Tampere)
- Marko Seppänen (Tampere University of Technology)

Global and Temporal Analysis of Social Positions at SourceForge.net

- Scott Christley (Dept. of Computer Science and Engineering, University of Notre Dame)
- Greg Madey (Dept. of Computer Science and Engineering, University of Notre Dame)

The user involvement process on open source e-learning tools

- Thiago Moreira (tjml@cin.ufpe.br)
- Alex Sandro Gomes (asg@cin.ufpe.br)
- Fábio Caparica (fcl@cin.ufpe.br)
- Rogério Nibon (rtn2@cin.ufpe.br)

Will Open Source Software Promise China a New Future of Domestic Software Industry?

- Yuping Song (Henan University of Technology School of Law, Zhengzhou, Henan, China 450001, songyup@gmail.com)

A Reference Model for F/OSS Process Management

- Michel Pawlak (University of Geneva, Switzerland, pawlak@cui.unige.ch)
- Ciaran Bryce (University of Geneva, Switzerland, Ciaran.Bryce@unige.ch)

Printed in the United States of America